TRAFFIC GROOMING IN SONET/WDM NETWORKS

by

Yong Wang

M.Sc., Simon Fraser University, 2003

B.Sc., Peking University, 1999

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Summer 2008

APPROVAL

Name:	Yong Wang
Degree:	Doctor of Philosophy
Title of thesis:	Traffic Grooming in SONET/WDM Networks
Examining Committee:	Dr. Ramesh Krishnamurti Chair
	Dr. Qian-Ping Gu, Senior Supervisor
	Dr. Joseph G. Peters, Supervisor
	Dr. Jiangchuan Liu, SFU Examiner
	Dr. Kshirasagar (Sagar) Naik, External Examiner, Associate Professor, Department of Electrical and Computer Engineering, University of Waterloo
Date Approved:	August 5, 2008

Abstract

In SONET/WDM networks, the bandwidth requirement of an individual network traffic demand is normally much lower than the capacity provided by a wavelength channel. Therefore multiple low-rate traffic demands are usually multiplexed together to share a high-speed wavelength channel during the transmission, and demultiplexed when arriving at corresponding destinations. This multiplexing/demultiplexing is known as *traffic grooming* and carried out by SONET Add-Drop Multiplexers (SADM). Since SADMs are expensive network devices, optimization problems in traffic grooming have been focusing on making efficient use of the SADMs. Traffic grooming has attracted a lot of research attention, and the optimization problems are challenging and NP-hard for almost all possible problem settings. In this thesis, we will study the traffic grooming problem and focus on designing efficient performance guaranteed algorithms for Unidirectional Path-Switch Ring (UPSR) networks in the following three categories: Firstly, we study the traffic grooming problem to minimize the total number of required SADMs in order to satisfy a given set of traffic demands, and aim to get better upper bounds on the number of SADMs than those achieved by previous algorithms. Secondly, we analyze the computational complexity and propose an efficient approximation algorithm for grooming the regular traffic pattern, which has not been studied previously. Thirdly, we study the computational complexity and propose efficient approximation algorithms for the Min-Max traffic grooming problem and the Maximum Throughput traffic grooming problem. We will also study the traffic grooming problems in Bidirectional Line-Switched Ring (BLSR) networks and discuss the extensions of our results for UPSR networks to BLSR networks. Finally, we will survey existing research problems on traffic grooming in other network topologies, for which we will discuss possible future research directions.

To my beloved mother

Acknowledgments

I would like to express my deepest thanks to my senior supervisor, Dr. Qian-ping Gu, for his valuable guidance, continuous encouragement, and instructions on academic writing throughout my Ph.D. research. I am very thankful to my supervisor, Dr. Joseph G. Peters, who is the organizer of Network Modeling Group. I learned much through our group meetings. I would also like to thank Dr. Jiangchuan Liu for serving as the internal examiner, Dr. Kshirasagar Naik for serving as the external examiner, and Dr. Ramesh Krishnamurti for serving as the chair of the examining committee.

I would like to take this opportunity to say thanks to all the friends whom I have met at Simon Fraser University, and all the staff and faculty in the School of Computing Science, for making a nice study and working environment.

Last but not least, I am deeply in debt to my parents for their continuous love. I would like to express my great gratitude to my wife for her love and encouragement all the time.

Contents

List of Figures

Chapter 1

Overview

1.1 Background

1.1.1 WDM technology

With the explosive growth of the Internet traffic, and the increasing demand for delay-sensitive multimedia network applications, it has been a challenging issue for the telecommunication infrastructure to provide sufficient bandwidth capacity for numerous Internet users. The introduction of fiber optics brings a bandwidth revolution to solve this problem. Optical fibers provide huge bandwidth capacity, which can be up to 50 terabits per second. However, the electronic network end devices usually operate on a speed of a few gigabits per second [45]. To eliminate the huge opto-electronic bandwidth mismatch, the bandwidth of an optical fiber is split into a number of non-overlapping wavelength channels, each of which operates at an electronic data rate. This multiplexing is known as the Wavelength Division Multiplexing (WDM) technology, which is widely used in optical networks to exploit the tremendous bandwidth capacity inherent in optical fibers.

1.1.2 Traffic grooming

In a WDM optical network, every optical fiber supports multiple wavelength channels, each of which has a bandwidth up to a few gigabit per second. However, the bandwidth requirement of an individual network traffic demand is normally much lower than the capacity provided by a wavelength channel. Therefore multiple low-rate traffic demands are

usually multiplexed together to share a high-speed wavelength channel during the transmission, and demultiplexed when arriving at corresponding destinations. The multiplexing/demultiplexing is known as *traffic grooming*, and the maximum number of traffic demands that can be multiplexed into a wavelength channel is called *grooming factor*. For example, four OC-3 (155.52 Mbps) traffic demands can be multiplexed into a wavelength channel operated at OC-12 (622.08 Mbps), giving a grooming factor of 4.

In SONET/WDM networks, traffic grooming is carried out by SONET Add-Drop Multiplexers (SADM), where one SADM is used to multiplex/demultiplex the traffic demands into/from a specific wavelength channel at each network node. With the emerging optical devices such as Wavelength Add-Drop Multiplexers (WADM), it is possible for a network node to optically bypass a wavelength channel if the wavelength does not carry any traffic ending at the node. Therefore, at each network node SADMs are needed only for the wavelengths which carry traffic from/to the node. Figure 1.1 shows the difference between an optical network node architecture without WADM (Figure 1.1(a)) and a node architecture with WADM (Figure 1.1(b)), where it is assumed that only wavelength λ_2 carries low-rate traffic demands from/to the network node shown in the Figure. It is clear that the number of SADMs can be decreased by deploying one WADM at each network node, and arranging the multiplexing/demultiplexing carefully. SADMs are expensive network devices and dominate the cost of SONET/WDM networks, so it is critical to minimize the number of SADMs in the SONET/WDM network design by utilizing the optical bypass capability of WADMs. Generally speaking, to reduce the number of SADMs, low-rate traffic demands should be groomed in a way to yield as many optical bypasses as possible.

In the following, we use a simple example to illustrate that a good grooming scheme saves the number of used SADMs. Consider a unidirectional ring network of four nodes shown in Figure 1.2 (a), where traffic demands must be routed in the clockwise direction and each optical fiber supports two wavelength channels. Suppose that each of the two wavelengths λ_1, λ_2 operates with a bandwidth OC-48, and there is an OC-16 traffic demand between each pair of nodes in the network. Since the grooming factor is 3 and every traffic demand is routed in the clockwise direction, a possible grooming scheme is as follows: traffic demands $(1,2),(2,1),(2,3),(3,2),(3,4),(4,3)$ are groomed into wavelength λ_1, and traffic demands $(1,3),(3,1),(2,4),(4,2),(1,4),(4,1)$ are groomed into wavelength λ_2. This scheme requires eight SADMs in total, which are listed in details as follows (see Figure 1.2 (b)):

1. Node 1 need one SADM for wavelength λ_1 to add traffic $(1,2)$ and drop traffic $(2,1)$;

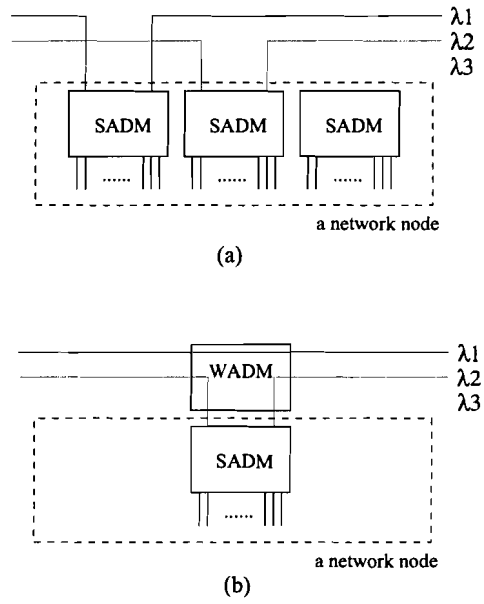

Figure 1.1: Node architecture without/with WADM.

2. Node 1 need one SADM for λ_2 to add traffic $(1,3)$, $(1,4)$ and drop traffic $(3,1)$, $(4,1)$);
3. Node 2 need one SADM for λ_1 to add traffic $(2,1)$, $(2,3)$ and drop traffic $(1,2)$ $(3,2)$;
4. Node 2 need one SADM for λ_2 to add traffic $(2,4)$ and drop traffic $(4,2)$;
5. Node 3 need one SADM for λ_1 to add traffic $(3,2)$, $(3,4)$ and drop traffic $(2,3)$, $(4,3)$;
6. Node 3 need one SADM for λ_2 to add traffic $(3,1)$ and drop traffic $(1,3)$;
7. Node 4 need one SADM for λ_1 to add traffic $(4,3)$ and drop traffic $(3,4)$;
8. Node 4 need one SADM for λ_2 to add traffic $(4,2)$, $(4,1)$ and drop traffic $(2,4)$, $(1,4)$.

A better grooming scheme is as follows: $(1,2), (2,1), (2,4), (4,2), (1,4), (4,1)$ are groomed into wavelength λ_1, and $(1,3), (3,1), (2,3), (3,2), (3,4), (4,3)$ are groomed into wavelength λ_2. This scheme requires seven SADMs in total, where two SADMs for each of node 1, 2, 4 and one SADM for node 3 (see Figure 1.2 (c)).

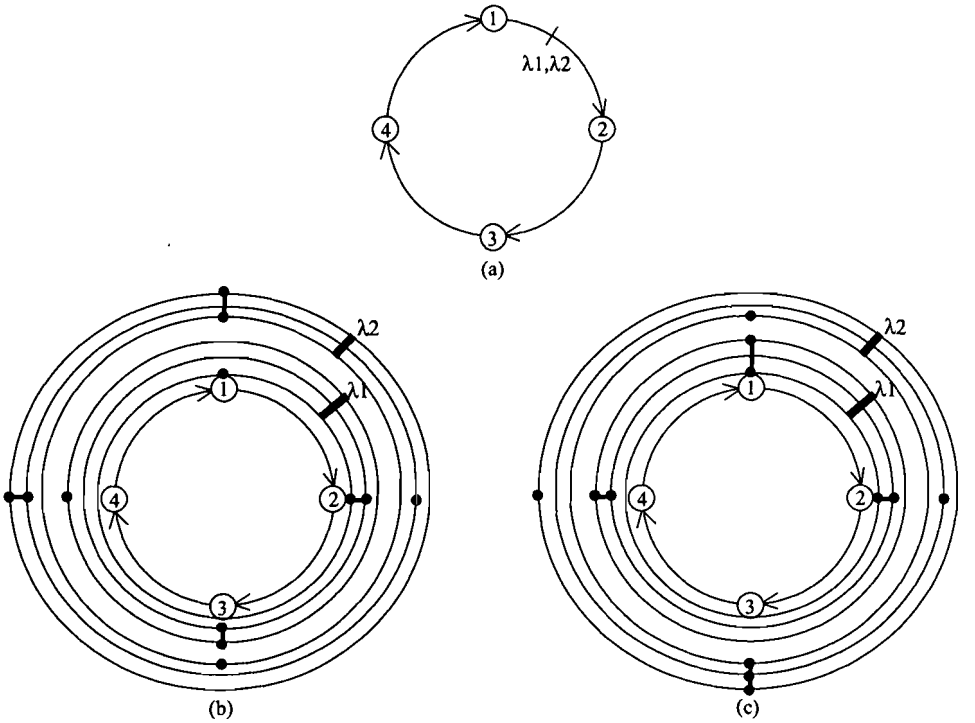

Figure 1.2: A traffic grooming example.

A major objective of traffic grooming in SONET/WDM networks is to find a grooming scheme for a given set of traffic demands, such that the total number of required SADMs is minimized. This problem is surveyed in [20, 43, 47, 68]. Also, it is worth pointing out that the traffic grooming problem is closely related to another important optimization problem in optical network design, which is to minimize the number of required wavelengths in order to satisfy a given set of traffic demands. Intuitively it seems that a smaller number of wavelengths implied a smaller number of SADMs. However, previous studies have shown that the two optimization goals can not be achieved simultaneously in general [7, 25, 42]. Therefore, it is natural to optimize one goal subject to a specific constraint of the other goal. Specifically, grooming schemes for minimizing SADMs subject to using the minimum number of wavelengths have been explored in [7, 32, 42, 64].

Another important optimization goal for the traffic grooming problem is to minimize

the maximum number of SADMs at a network node over all network nodes (known as the Min-Max traffic grooming problem in [12]). Such a Min-Max traffic grooming problem is of high practical interest. First of all, minimizing the maximum number of SADMs very likely produces a homogeneous network design in terms of the number of SADMs at each network node. Homogeneous nodes usually have lower cost than heterogeneous ones for network deployment, maintenance and upgrade due to the identical specifications. Furthermore, the Min-Max optimization goal is important for dealing with dynamic traffic demands. In particular, if the traffic pattern is changing dynamically within a given set, we can pre-compute a solution for each traffic pattern in the set, and deploy in every node with the largest number of SADMs required by any solution. Thus any traffic pattern from the given set can be satisfied without reconfiguring the deployment of SADMs.

It is also an important optimization goal for the traffic grooming problem to maximize the number of accommodated traffic demands subject to using limited traffic grooming resources (known as the Maximum Throughput traffic grooming problem in [67, 66, 65]). This problem is critical in the situation that there are no sufficient SADMs in the network to satisfy the full connectivity of the traffic demands. Especially in the WDM network operation, when the deployed SADMs are not sufficient to satisfy the full connectivity for a given set of traffic demands, a grooming scheme which satisfies as many traffic demands as possible is greatly desired.

1.1.3 Network topology

The traffic grooming problem has attracted a lot of research attention, and previous work in the literature is mainly in ring networks. SONET rings have been widely used to construct Metropolitan Area Networks (MAN), which are typically backbone optical networks that span metropolitan areas. For example, the telephone companies have provided MAN services in the form of SONET rings for years [1]. We will give a brief introduction on the SONET ring topology in the following.

There are two main architectures in SONET/WDM ring networks: Unidirectional Path-Switched Ring (UPSR) networks and Bidirectional Line-Switched Ring (BLSR/2 or BLSR/4) networks [26]. In a UPSR network, there is a pair of optical fibers between each pair of adjacent nodes in the ring. Each fiber provides a unidirectional link and the two fibers work in opposite directions. The fibers in a UPSR constitute two unidirectional rings with one in the clockwise direction and the other in the counter-clockwise direction, where one ring is

used as a working ring and the other as a protecting ring (see Figure 1.3 (a)). A bidirectional communication between two nodes in a UPSR is realized with two nodes sending traffic on the working ring, for example, both in the clockwise direction (i.e., all the fibers in the working ring are used). In a BLSR/2 network, all fibers also constitute two unidirectional rings that are the same as those in a UPSR network. However, the two unidirectional fiber rings are both used as working rings, and the protection is provided by reserving 50% of the capacity in each ring (see Figure 1.3 (b)). In a BLSR/4 network, there are two extra unidirectional rings (one in the clockwise direction and the other in the counter-clockwise direction) compared to BLSR/2. The protection is provided by reserving all the capacity of the two extra rings (see Figure 1.3 (c)). A bidirectional communication between two nodes in a BLSR may be realized with one node sending traffic in the clockwise direction and the other node sending traffic in the counter-clockwise direction (i.e., both working rings are partly used). Compared to UPSR networks, the routing in BLSR networks is not unique, which makes the traffic grooming problem in BLSR more complicated.

1.2 Previous work

Previous work on traffic grooming in UPSR and BLSR networks can be classified into two categories: approximation algorithms with provable worst case performance guarantees, and heuristic algorithm evaluated by empirical results. Traffic grooming is a very complicated optimization problem, and usually it is difficult to obtain approximation algorithms with guaranteed performances. In previous work, the performance guaranteed algorithms are available only for all-to-all traffic pattern and duplex traffic pattern in UPSR networks [42, 32, 64, 7, 28, 11], where the optimization goal is to minimize the total number of required SADMs to satisfy a given set of traffic demands. For the arbitrary traffic grooming in UPSR networks, and traffic grooming in BLSR networks, various heuristic algorithms have been proposed [52, 50, 16, 17, 51].

1.3 Thesis contributions

In this thesis, we will focus on developing performance guaranteed algorithms for traffic grooming problems. We will propose performance guaranteed algorithms for UPSR networks in the following three categories.

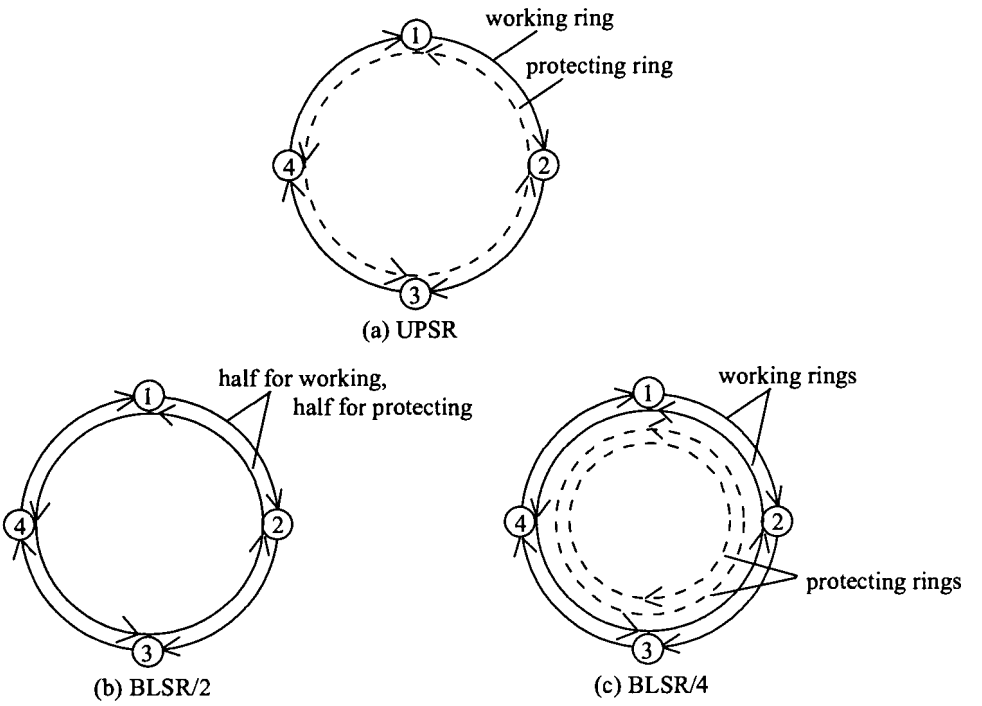

Figure 1.3: Optical ring topologies.

First, we study the traffic grooming problem to minimize the total number of SADMs [55, 58]. We will propose two linear time approximation algorithms, which achieve better worst case performance guarantees than previous algorithms. In addition, both of our algorithms use the minimum number of wavelengths. Second, we consider the traffic grooming problem with regular traffic pattern [56, 57], which has not been studied previously. We prove the problem is NP-hard, and show that the problem does not admit a Fully Polynomial Time Approximation Scheme (FPTAS) unless $P = NP$. We further prove that the problem remains NP-hard even if the grooming factor is any value chosen from a subset of integers. We also propose an approximation algorithm to solve the problem. As well, the proposed algorithm always uses the minimum number of wavelengths. Last but not least, we study the Min-Max traffic grooming problem [60] and the Maximum Throughput traffic grooming problem [59]. For each of the two problems, we prove the NP-hardness of the problem,

and propose an approximation algorithm. We also study the all-to-all traffic pattern, and present algorithms achieving solutions only constant factors away from the optimal ones.

In addition, we will study the traffic grooming problem in BLSR networks. As the topology changes from UPSR to BLSR, the complexity of the traffic grooming problem increases. There is no performance guaranteed algorithm available so far. We will extend our algorithms proposed for UPSR networks to design efficient heuristic algorithms for traffic grooming in BLSR networks.

1.4 Thesis organization

This thesis is organized as follows: In Chapter 2, we first present preliminaries, and then summarize the previous work that is closely related to our research. In Chapter 3, we propose algorithms for the traffic grooming problem to minimize the total number of required SADMs in UPSR networks, and obtain improved guaranteed performance than previous algorithms. In Chapter 4 we study traffic grooming in UPSR networks with regular traffic pattern, which has not be considered before. In Chapter 5 we consider the Min-Max traffic grooming problem and the Maximum Throughput traffic grooming problem in UPSR networks. We analyze the computational complexity for both problems, and propose performance guaranteed algorithms for them. In Chapter 6, we study traffic grooming in BLSR networks. Future extensions to traffic grooming in other network topologies are discussed in Chapter 7. The final chapter concludes the thesis.

Chapter 2

Preliminaries and related work

In this chapter, first we will provide the notation and terminology, which will be used throughout the thesis. Then we will survey the existing research of traffic grooming in the literature. In particular, my main focus will be on the computational complexity of the traffic grooming problems, and the performance guaranteed algorithms.

2.1 Preliminaries

2.1.1 Graph theory notation and terminology

In this thesis, we will propose performance guaranteed algorithms for traffic grooming using a novel graph partitioning approach. We use the nodes in a graph to denote the nodes in the network, and use the edges to model the traffic demands between network nodes. We will first introduce the graph theory notation and terminology used throughout the thesis in the following. Readers are referred to a textbook on graph theory (e.g., the one by West [61]) for basic definitions and terminology on graphs.

Let $G(V, E)$ be a simple connected undirected graph with node set $V(G)$ and edge set $E(G)$. For $v \in V(G)$, the degree $\delta(v)$ of v is the number of edges of G incident to v. If $\delta(v) = r$ for every $v \in V(G)$, we say G is a *r-regular graph*. We use $\Delta(G)$ to denote the maximum degree over all nodes in G, (i.e., $\Delta(G) = \max_{v \in V(G)}\{\delta(v)\}$), and we simply use Δ instead of $\Delta(G)$ when it is clear from the context. A *path* of G is a sequence of consecutive edges $\{x_0, x_1\}$,$\{x_1, x_2\}$,...,$\{x_{l-1}, x_l\}$ in G, where no repeated edge is allowed in the path, and we use $x_0 - x_1 - x_2 - \cdots - x_{l-1} - x_l$ to denote the path for simplicity. The *length* of

a path is the number of edges in the path, and we use *l-path* to denote a path of length l. The two nodes at which a path starts and terminates are called *end-nodes* of the path, and all other nodes on the path are called *mid-nodes*. A *simple path* is a path with no repeated node. An *Euler path* of graph G is a path which uses each edge of G exactly once, and a connected graph has an Euler path if and only if it has at most two odd-degree nodes. A *component* of G is a maximal connected sub-graph of G. The *edge connectivity* of G is defined to be the minimum number of edges whose deletion from G partitions G into at least two components.

A *tree* T is a connected graph with $|V(T)| - 1$ edges, and a *spanning tree* of G is a tree that contains all nodes of G. A *rooted spanning tree* of G is a spanning tree of G where a node r is specified as the *root*. For any node u other than root r in a rooted spanning tree T, the *parent* of u, denoted as u_p, is the node adjacent to u on the path from u to r in T. For any node u in T, a *descendant* of u is a node x such that u is on the unique path from x to r, and the length of the unique path from x to u in tree T is denoted by $d_T(x, u)$. For any node u in T, the *depth* of u is defined as $d_T(u, r)$. For a node u in T, the subtree induced by node u and all its descendants is denoted by T_u, and node u is the root of T_u. The depth of T_u is defined as $d_T(u, r)$ (i.e., the depth of node u). The *height* of T_u, denoted as $h(T_u)$, is defined as $\max\{d_T(x, u) | x \in V(T_u)\}$. A tree is called *trivial* if the tree contains only one node.

For a simple undirected graph $G(V, E)$, a *matching* M of G is a set of edges of G such that no two edges of M share a node in common. We say a node is *saturated* by matching M if the node is an end-node of some edge in M. If a matching saturates every node of G, then it is a *perfect matching*. For graph G and a function $b : V(G) \rightarrow \mathcal{Z}^+$, a *b-matching* M is a set of edges in G such that each node v of G appears in at most $b(v)$ edges of M. A maximum b-matching is a b-matching with the maximum cardinality, and can be computed in polynomial time for a simple graph G [23]. An *edge coloring* of G is a coloring of the edges of G such that adjacent edges receive different colors. For graph G and a function $f : V(G) \rightarrow \mathcal{Z}^+$, an *f-coloring* is to assign a color for each edge in $E(G)$, such that at most $f(v)$ edges incident to a node v receive the same color. It is proved that an f-coloring with at most $\Delta_f + 1$ colors can be computed in polynomial time for a simple graph G, where $\Delta_f = \max_{v \in V(G)} \lceil \delta(v)/f(v) \rceil$ [30]. The edge coloring is a special case of the f-coloring, where $f(v) = 1$ for each $v \in V(G)$.

A *clique* of graph G is a complete sub-graph of G, and the *size* of the clique is the number

of nodes in the clique. A *star*, denoted as S, is a tree with one node having degree $|V(S)|-1$ and the others having degree 1. The node of degree $|V(S)|-1$ is called the *center* of star S. A *bipartite graph* $G(U,V,E)$ is a graph whose node set can be divided into two non-empty sets $U(G)$ and $V(G)$ such that for every edge $\{u,v\} \in E(G)$, one of nodes u,v is in $U(G)$ and the other is in $V(G)$.

2.1.2 Approximation algorithm

For any NP-hard optimization problem, there is no polynomial time algorithm to compute an optimal solution unless $P = NP$. So much effort has been put into developing polynomial time algorithms to find near-optimal solutions whose values are close to the value of an optimal solution. We call such a near-optimal solution an *approximate solution*, and an algorithm that produces approximate solutions an *approximation algorithm*. The following formal definitions of the approximate solution, approximation algorithm, and approximation ratio are from the book by Cormen et al. [18].

Approximate Solution: for an optimization problem $\prod$*, a feasible solution with the value close to the value of an optimal solution is an approximate solution of* $\prod$.

Approximation Algorithm: for an optimization problem $\prod$*, a polynomial time algorithm that generates approximate solutions is an approximation algorithm of* $\prod$.

Any algorithm that produces approximate solutions for problem $\prod$ can be called an approximation algorithm. How to evaluate approximation algorithms is important. Generally speaking, good approximation algorithms should be efficient (polynomial time) and produce solutions with values as close to the optimum as possible. Concept *approximation ratio* is widely used to evaluate approximation algorithms. In the following, we assume that a solution always has a positive value.

Approximation Ratio: an approximation algorithm A for an optimization problem $\prod$ *has an approximation ratio of* $\rho(n)$ *if for any instance of* $\prod$ *with size* n*, the value* C *of any approximate solution produced by the approximation algorithm satisfies*

$$\max\{\frac{C}{C^*}, \frac{C^*}{C}\} \leq \rho(n),$$

where C^* is the value of an optimal solution.

This definition applies to both maximization and minimization problems. For a maximization problem, $0 < C \leq C^*$, and C^*/C gives the approximation ratio. For a minimization problem, $0 < C^* \leq C$, and C/C^* gives the approximation ratio. Therefore, the approximation ratio of an approximation algorithm is at least 1, and the approximation ratio of an optimal algorithm is exactly 1. An approximation algorithm with a large approximation ratio may return a solution that is much worse than an optimal solution. Reducing the approximation ratio as close to 1 as possible is a major goal of developing approximation algorithms.

For a minimization problem $\prod$, an algorithm A is called a *Polynomial Time Approximation Scheme (PTAS)* if its running time, for any instance of $\prod$ and a parameter of fixed $\epsilon > 0$, is bounded by a polynomial in the size of the instance, and the value C of any solution produced by A satisfies $\frac{C}{C^*} \leq 1 + \epsilon$, where C^* is the value of an optimal solution. A *Fully Polynomial Time Approximation Scheme (FPTAS)* is a PTAS for which the running time is bounded polynomially in both the size of the problem instance and $\frac{1}{\epsilon}$.

2.2 Related work

2.2.1 Traffic grooming in UPSR networks

Traffic grooming is a complicated optimization problem, and usually it is difficult to obtain polynomial time optimal algorithms or even approximation algorithms with guaranteed performances. So far, the performance guaranteed algorithms are available only for all-to-all traffic pattern and duplex traffic pattern.

All-to-all traffic pattern

For the all-to-all traffic pattern, there is a traffic demand (x, y) from node x to node y for every pair of nodes x and y in the UPSR network. A traffic demand is *unitary* if it requires one unit of bandwidth. Without loss of generality, we assume that every traffic demand is unitary, since otherwise we can split a traffic into multiple unitary sub-traffic. We call (x, y) and (y, x) a *duplex traffic pair*, and use $\{x, y\}$ to denote the unitary duplex traffic pair (x, y) and (y, x). In the UPSR network, every duplex pair of unitary traffic occupies a full

cycle with one unit of bandwidth. Such a full cycle is called a *primitive cycle*, and up to k primitive cycles can be groomed into one wavelength for a grooming factor of k.

For the all-to-all traffic grooming with grooming factor $k = 4$ and $k = 16$, Chiu and Modiano [42] derived lower bounds on the number of SADMs, and Hu [32] provided the optimal solutions. For an arbitrary grooming factor k, lower bounds and heuristics were proposed by Chiu and Modiano [42], and Zhang and Qiao [64], where the heuristics are evaluated by computer simulations, yet their approximation ratios are not given.

By formulating the traffic grooming problem as a graph partitioning problem, Bermond and Coudert [7] optimally solved the all-to-all traffic grooming problem in UPSR networks if the grooming factor k is a practical value or in the infinite congruence classes of values. The graph partitioning formulation is based on the following fact which has been assumed implicitly in [7]: for any grooming scheme $\mathcal{G}$, a grooming scheme $\mathcal{G}'$ can be constructed such that $\mathcal{G}'$ always puts each unitary duplex pair into one primitive cycle and $\mathcal{G}'$ uses no more SADMs than $\mathcal{G}$ (we will give the explicit proof for this claim in the next chapter). Therefore for the all-to-all traffic pattern, we can always put each duplex pair into a primitive cycle, and concentrate on grooming primitive cycles into wavelengths so that the number of used SADMs is minimized. In the graph partitioning approach proposed by [7], the all-to-all traffic pattern is represented by an undirected complete graph $G(V, E)$ called *traffic graph*, where $V(G)$ represents the set of nodes in the UPSR network and each edge in $E(G)$ between node x and y represents the duplex traffic pair $\{x, y\}$. Then the all-to-all traffic grooming problem is formulated as the following k-Edge-Partitioning problem of the traffic graph:

For a positive integer $k \leq |E(G)|$, partition the edge set $E(G)$ into a collection of subsets $\mathcal{E} = \{E_1, E_2, \ldots, E_W\}$ (where $\bigcup_{i=1}^{W} E_i = E(G)$ and $E_p \cap E_q = \emptyset$ for $p \neq q$), such that $|E_i| \leq k$ for each $E_i \in \mathcal{E}$ and $\sum_{E_i \in \mathcal{E}} |V_i|$ is minimized, where V_i is the set of nodes in the sub-graph induced by edge set E_i.

It is observed that integer k corresponds to the grooming factor, W corresponds to the number of used wavelengths, each induced sub-graph corresponds to a wavelength, and $\sum_{E_i \in \mathcal{E}} |V_i|$ corresponds to the total number of used SADMs. We will take the example given in Section 1.1.2 to illustrate the graph partitioning approach. The all-to-all traffic pattern in the example can be represented by a complete graph K_4 with 4 nodes shown in Figure 2.1 (a). The grooming scheme that uses 8 SADMs corresponds to the graph partition of K_4

shown in Figure 2.1 (b), and the grooming scheme that uses 7 SADMs corresponds to the graph partition shown in Figure 2.1 (c).

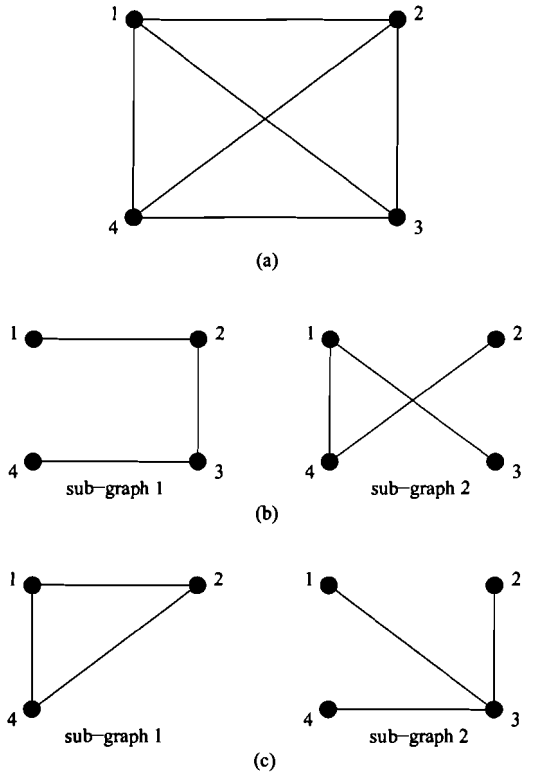

Figure 2.1: A graph partitioning example.

Bermond and Coudert [7] tackled the graph partitioning problem using the techniques of design theory [15]. Their results improve and unify all the previous results on the all-to-all traffic grooming in UPSR networks. In particular, they completely or partially solved the cases that $k = 3, 4, 5, 6, 8, 9, 10, 16$ and various congruence classes for other values of k. We will show the case $k = 3$ as an example of their results in the following. By using the results from design theory, Bermond and Coudert proved the following result, where $E, P_3, K_3, K_{1,3}, P_4$ denote the graphs shown in Figure 2.2:

For complete graph K_n with n nodes, the optimal k-edge partition of K_n for $k = 3$ is as

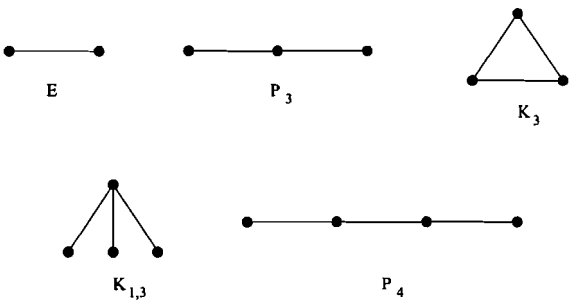

Figure 2.2: Examples of graphes with at most three edges.

follows:

1. $\frac{n(n-1)}{6}$ K_3 *when* $n \equiv 1, 3 (mod\ 6)$;
2. $\frac{n(n-1)-8}{6}$ K_3 *and* 2 P_3 *when* $n \equiv 5 (mod\ 6)$;
3. $(\frac{n(n-1)}{6} - \frac{n}{4})$ K_3 *and* $\frac{n}{4}$ $K_{1,3}$ *when* $n \equiv 0, 4 (mod\ 12)$;
4. $(\frac{n(n-1)-2}{6} - \lceil \frac{n-2}{4} \rceil)$ K_3, $\lceil \frac{n-2}{4} \rceil$ $K_{1,3}$ *and* 1 E *when* $n \equiv 2, 8 (mod\ 12)$;
5. $(\frac{n(n-1)}{6} - \frac{n+2}{4})$ K_3, $\frac{n-2}{4}$ $K_{1,3}$ *and* 1 P_4 *when* $n \equiv 6, 10 (mod\ 12)$.

Bermond and Coudert [7] also discussed minimizing the number of used wavelengths, and provided a deep insight into the cases for which the number of SADMs and the number of wavelengths can not be minimized simultaneously.

Duplex traffic pattern

A set R of traffic demands is *duplex* if $(x, y) \in R$ implies $(y, x) \in R$, where (x, y) denotes a traffic demand from node x to node y. Duplex traffic demands are very common in many applications, for example, IP telephony. The all-to-all traffic pattern discussed in the previous section is a special case of the duplex traffic pattern. For the duplex traffic grooming, we can similarly put each duplex pair into a primitive cycle, and concentrate on grooming primitive cycles into wavelengths so that the number of used SADMs is minimized. Therefore, the graph partitioning approach discussed for the all-to-all traffic pattern in the previous section can be used for the duplex traffic pattern as well: the set R of traffic

demand pairs is represented by an undirected graph $G(V, E)$, where $V(G)$ represents the set of nodes in the UPSR network, and there is an edge in $E(G)$ between node x and y if and only if $\{x, y\} \in R$. The only difference from the all-to-all traffic grooming is that we now consider the k-Edge-Partitioning problem on arbitrary traffic graphs rather than complete traffic graphs.

For arbitrary traffic graphs, Goldschmidt *et al.* [28] proved that the k-Edge-Partitioning problem is NP-hard. The NP-hardness proof is based on the reduction from the Edge-Partition into Triangles (EPT) problem, which is known to be NP-hard [31] and the decision version of the EPT problem is stated as follows:

Edge-Partition into Triangles (EPT) Problem

Instance: An undirected graph $G(V, E)$.

Question: Is there a partition of $E(G)$ into pairwise disjoint subsets $E_1, E_2, ..., E_{|E(G)|/3}$, such that each E_i induces a triangle?

For the NP-completeness proof, it is shown that any given instance of the EPT problem can be transformed to an instance of the k-Edge-Partitioning problem on the same graph with $k = 3$, such that the instance has a solution if and only if the original EPT instance has a solution.

Goldschmidt *et al.* [28] also proposed an approximation algorithm for the k-Edge-Partitioning problem on arbitrary traffic graphs. The algorithm first transforms a traffic graph G into an open tree as follows: it computes a spanning tree T of G. Then for every edge $\{u, v\} \in E(G) \backslash E(T)$, it re-labels node v as a new node and attaches edge $\{u, v\}$ to node u in T. Figure 2.3 illustrates how to construct an open tree for a given graph G. It is clear that an open tree of G has the same number of edges as G, and has duplicated copies for some nodes of G. Then the open tree is partitioned into subtrees such that $\lceil \frac{k}{2} \rceil \leq |E(T')| \leq k$ for each subtree T'. For each subtree, a sub-graph G_i of G can be obtained by merging duplicated copies of the same node in the subtree. All such sub-graphs together constitute a k-edge partition of G. It is clear that the number of edges $|E(G_i)|$ in each sub-graph G_i satisfies $\lceil \frac{k}{2} \rceil \leq |E(G_i)| \leq k$. For each sub-graph G_i, the number of nodes $|V(G_i)| \leq |E(G_i)| + 1$, where the equation holds if and only if G_i is a tree. Since each sub-graph contains at least $\lceil \frac{k}{2} \rceil$ edges, the total number of sub-graphs is at most $\frac{|E(G)|}{\lceil \frac{k}{2} \rceil} \leq \frac{2|E(G)|}{k}$. Therefore, the total number of nodes over all sub-graphs is $\sum_i |V(G_i)| \leq \sum_i (|E(G_i)| + 1) \leq |E(G)| + \frac{2|E(G)|}{k} = |E(G)|(1 + \frac{2}{k})$, which means the algorithm uses at most $\lceil |E(G)|(1 + \frac{2}{k}) \rceil$ SADMs.

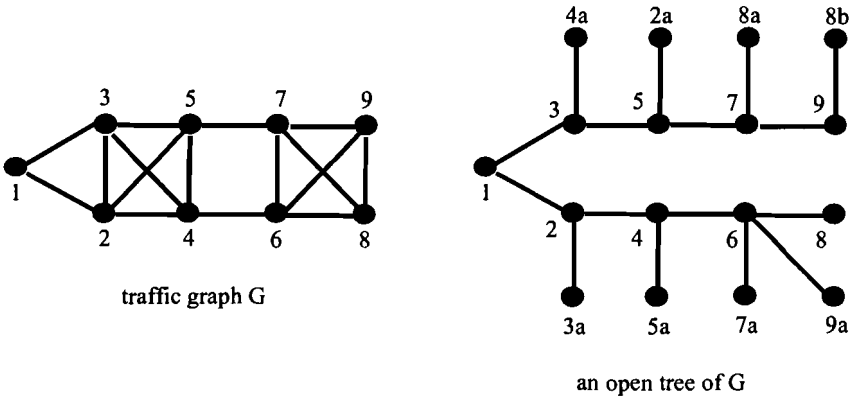

Figure 2.3: Illustration of constructing the open tree of a traffic graph.

Brauner *et al.* [11] proposed another two approximation algorithms for the k-Edge-Partitioning problem on arbitrary traffic graphs. The first algorithm is based on an approach that constructs edge-disjoint 2-paths of traffic graph G [41]. Then the set of 2-paths are partitioned into subsets, each of which contains $\lfloor \frac{k}{2} \rfloor$ 2-paths. The edges appearing in the 2-paths of each subset induce a sub-graph of G, and all sub-graphs constitute a k-edge partition of G. It can be shown that this algorithm uses at most $\lceil \frac{3|E(G)|}{2} \rceil$ SADMs. The second algorithm starts by adding virtual edges between odd-degree nodes to make every node of G have even degree. Then an Euler path is constructed, and cut into segments such that each segment contains k edges of the original graph G and some virtual edges. The k-edge partition is obtained by removing virtual edges and merging duplicated copies of the same node in each segment. This second algorithm uses at most $\lceil (1 + \frac{1}{k})|E(G)| \rceil + \frac{\Delta_{odd}}{2}$ SADMs, where Δ_{odd} is the number of odd-degree nodes in the graph.

Arbitrary traffic pattern

Besides the performance guaranteed algorithms for all-to-all traffic pattern and duplex traffic pattern, there are also heuristic algorithms proposed for arbitrary traffic pattern. We will show some representative results in the following.

Due to the hardness of the traffic grooming problem, it is difficult to develop performance guaranteed algorithms for the arbitrary traffic pattern. Instead, the heuristic algorithms

evaluated by simulations have been proposed by a lot of research papers, among which the work by Wan *et al.* [52] can be considered as a representative one.

For an arbitrary traffic pattern, the algorithm in [52] uses a two-step approach to solve the traffic grooming problem: generation of primitive cycles and grooming of primitive cycles. In the first step to generate primitive cycles, the set of traffic demands are partitioned into groups such that the routing paths for the traffic demands within any group do not overlap. Therefore, the traffic demands in each group can be put into one primitive cycle. The cost of each group (or primitive cycle) is defined as the number of different nodes appearing as the endpoints of the routing paths contained in the primitive cycle, and the cost of a partition is defined as the sum of the costs of the primitive cycles within this partition. The objective of the first step is then to find a valid partition of the arbitrary traffic set into groups such that the partition has the minimum cost.

In the second step to groom primitive cycles, primitive cycles are groomed into high-speed wavelength channels such that the number of primitive cycles in each wavelength channel is no more than the grooming factor k. The cost of each wavelength channel is defined as the number of different nodes appearing as the endpoints of the routing paths in the primitive cycles groomed in the wavelength channel, that is, the number of SADMs required for the wavelength. Therefore, the objective of the second step is to groom a set of primitive cycles such that the number of required SADMs is minimized. Wan *et al.* [52] give a performance guaranteed sub-algorithm for each of the two steps. However, there is no proof (and it is difficult to prove) on a set of solutions of the first step, based on which optimal solutions for the whole problem can be derived in the second step. Therefore, the worst case performance of the two-step approach in [52] is not guaranteed.

2.2.2 Traffic grooming in BLSR networks

For BLSR networks, usually a similar two-step approach as the one in [52] is used to solve the traffic grooming problem, where the first step is to generate primitive cycles and the second step is to groom primitive cycles into wavelength channels. Traffic grooming in BLSR networks is more complicated than traffic grooming in UPSR networks, and there is no performance guaranteed algorithms so far even for all-to-all traffic pattern. However, there has been research work which proposes performance guaranteed sub-algorithms for each of the two steps. We will briefly review such algorithms for both all-to-all traffic pattern and arbitrary traffic pattern in this section.

All-to-all traffic pattern

Wan [50] studied the all-to-all traffic grooming on BLSR networks for grooming factor $k = 1, 2,$ and 4. Colbourn and Wan [17], and Colbourn and Ling [16] solved the case $k = 8$ using the techniques of design theory [15]. All of these works use the same two-step approach to tackle the traffic grooming problem: the first step is to generate primitive cycles and the second step is to groom primitive cycles, where the goal for the first step is to minimize the number of primitive cycles, and the goal of the second step is to minimize the number of SADMs.

For the first step, Wan [50] proposed the an approach to achieve the minimum number of primitive cycles: when the number n of nodes in the BLSR network is even, $\frac{n^2}{4}$ primitive cycles are constructed, where $\lceil \frac{n^2}{8} \rceil$ primitive cycles are in the clockwise direction and the other $\lfloor \frac{n^2}{8} \rfloor$ ones are in the counterclockwise direction; when n is odd, $\frac{n^2-1}{4}$ primitive cycles are constructed, where $\lceil \frac{n^2-1}{8} \rceil$ primitive cycles are in each of the clockwise and counterclockwise directions. For the second step, they model the primitive cycles constructed in the first step as a complete graph with a loop on each node, and then formulate the primitive cycle grooming into a graph partitioning problem: partition the edges of the graph into sub-graphs, each of which contains at most k edges (where loops are counted as edges when n is odd, and counted as half-edges when n is even), such that the total SADM cost of all sub-graphs is minimized. Based on this graph partitioning approach, the cases $k = 1, 2, 4, 8$ are studied in [16, 17, 50] to achieve the minimum number of SADMs given the primitive cycles obtained in the first step.

It is worth pointing out that the two-step approach does not guarantee that the number of used SADMs is optimum even the optimal solution can be achieved for each step individually. This is because that the two steps are not completely independent, and the solution to the first step might affect how the second step can be solved optimally.

Arbitrary traffic pattern

The two-step approach is used for arbitrary traffic grooming in BLSR networks as well [51]. The first step is to generate primitive cycles and the second step is to groom primitive cycles obtained from the first step. Two versions of the problem are considered in [51], where the first version is that each traffic stream has a predetermined routing, and the second version is that the routing of each traffic stream is not given in advance. It is proved

that both versions are NP-hard for any fixed grooming factor k. For both versions, various performance guaranteed algorithms are developed to solve primitive cycle generation sub-problem or the primitive cycle grooming sub-problem individually. However, it is difficult to derive performance guaranteed results for the whole traffic grooming algorithm as well.

Chapter 3

Traffic Grooming in UPSR networks

Minimizing the total number of required SADMs, as a major optimization goal of the traffic grooming problem, has been shown very challenging. It is NP-hard even for Unidirectional Path-Switched Ring (UPSR) networks with unitary duplex traffic pattern. In this chapter, we propose two linear time approximation algorithms for this NP-hard problem based on a novel graph partitioning approach. Both algorithms achieve better worst case performances than previous algorithms. We show that the upper bounds obtained by our algorithms are very close to the lower bounds for some instances. In addition, both of our algorithms use the minimum number of wavelengths, which are precious resources as well in optical networks. We also conduct extensive simulations to evaluate the average performances of our algorithms.

3.1 Problem formulation

Recall that in UPSR networks, a pair of unitary duplex traffic occupies a full cycle with one unit of bandwidth. Such a full cycle is called a *primitive cycle*, and up to k primitive cycles can be groomed into one wavelength for a grooming factor of k. To illustrate the graph partition formulation more clearly, we prove the following theorem first.

Theorem 1 *For any grooming scheme $\mathcal{G}$, there exists a grooming scheme $\mathcal{G}'$ such that $\mathcal{G}'$ always puts every unitary duplex pair into a primitive cycle and $\mathcal{G}'$ uses no more SADMs*

than $\mathcal{G}$.

Proof: Recall that a primitive cycle denotes a full cycle with unitary bandwidth capacity in the SONET/WDM ring network, that is, a wavelength channel along the ring network consists of k primitive cycles for a grooming factor of k. In the UPSR network, a primitive cycle can carry exactly one unitary duplex traffic pair. We consider any grooming scheme $\mathcal{G}$ as the following two-step procedure: the first step is a *packing scheme* $\mathcal{P}$ that puts traffic into primitive cycles, and the second step is a *clustering scheme* $\mathcal{C}$ that puts primitive cycles into wavelengths. We use $\mathcal{P}'$ to denote the special packing scheme that puts each duplex traffic pair into a primitive cycle. For any grooming scheme $\mathcal{G}$ consisting of a packing scheme $\mathcal{P}$ and a clustering scheme $\mathcal{C}$, if $\mathcal{P} = \mathcal{P}'$, $\mathcal{G}'$ can be constructed such that $\mathcal{G}'$=$\mathcal{G}$. Otherwise, it is observed that $\mathcal{P}'$ uses no more primitive cycles than $\mathcal{P}$. We call traffic demand (i, j) a *long demand* if its routing path in the UPSR has length at least $\frac{n}{2}$ (where n is the number of nodes in the UPSR), and a *short demand* otherwise. It is noticed that a duplex pair contains at least one long demand. Now we define the following function f to map primitives cycles used in $\mathcal{P}'$ to those used in $\mathcal{P}$:

1. For each primitive cycle x used in $\mathcal{P}'$ that carries a long demand (i, j) and a short demand (j, i), and suppose y is the primitive cycle used in $\mathcal{P}$ that carries (i, j), we define $f(x) = y$.

2. For each primitive cycle x used in $\mathcal{P}'$ that carries two long demands (i, j) and (j, i) (in this case, n is even, and the length of the routing paths for both (i, j) and (j, i) is $\frac{n}{2}$), and suppose y_1 is the primitive cycle used in $\mathcal{P}$ carrying (i, j) and y_2 is the primitive cycle used in $\mathcal{P}$ carrying (j, i), we define either $f(x) = y_1$ or $f(x) = y_2$.

The above function f is an injective function, since no primitive cycle can carry more than one long demand from different duplex traffic pairs. Now we construct grooming scheme $\mathcal{G}'$ with packing scheme $\mathcal{P}'$, and clustering scheme $\mathcal{C}'$ as follows: two primitive cycles x_1 and x_2 used in $\mathcal{P}'$ are put into the same wavelength if and only if $f(x_1)$ and $f(x_2)$ are put into the same wavelength in $\mathcal{C}$. Since f is an injective function, it is clear $\mathcal{G}'$ uses no more SADMs than $\mathcal{G}$. □

According to Theorem 1, for a set R of unitary duplex traffic demand pairs in UPSR networks, we can always put each pair into a primitive cycle, and concentrate on grooming primitive cycles into wavelengths so that the number of required SADMs is minimized. That

is, the traffic grooming problem can be solved by partitioning R into subsets, each of which contains at most k duplex pairs, and use one wavelength to carry each subset. For each node involved in at least one duplex pair of a subset carried by a wavelength λ, we need one SADM for wavelength λ at the node, and minimizing the total number of required SADMs is equivalent to minimizing the sum of the number of distinct nodes involved in each subset.

As we mentioned in Chapter 2, the traffic grooming problem in UPSR networks with unitary duplex traffic pattern has been formulated as the following k-Edge-Partitioning problem to develop approximation algorithms [11, 28]:

k-Edge-Partitioning Problem

Instance: A traffic graph $G(V, E)$ and integer $k \leq |E(G)|$.

Objective: Partition the edge set $E(G)$ into a collection of subsets $\mathcal{E} = \{E_1, E_2, \ldots, E_W\}$ (where $\bigcup_{i=1}^{W} E_i = E(G)$ and $E_p \cap E_q = \emptyset$ for $p \neq q$), such that $|E_i| \leq k$ for each $E_i \in \mathcal{E}$ and $\sum_{E_i \in \mathcal{E}} |V_i|$ is minimized, where V_i is the set of nodes in the sub-graph induced by edge set E_i.

In this formulation, a simple undirected graph $G(V, E)$, called *traffic graph*, is constructed to represent the set R of unitary duplex traffic demand pairs, where node set $V(G)$ denotes the set of network nodes in the UPSR and there is an edge $\{x, y\} \in E(G)$ between node x and y if and only if there exists a unitary duplex pair $\{x, y\} \in R$ (when it is clear in the context, we use $\{x, y\}$ to denote either an edge in the graph or a unitary duplex pair in R). It is observed that integer k corresponds to the grooming factor, W corresponds to the number of required wavelengths, and $\sum_{E_i \in \mathcal{E}} |V_i|$ corresponds to the total number of required SADMs. We also notice that $\left\lceil \frac{|E(G)|}{k} \right\rceil$ is a lower bound on the number of required wavelengths (i.e., $\left\lceil \frac{|E(G)|}{k} \right\rceil \leq W$).

3.2 Algorithms

For arbitrary traffic graphs, the k-Edge-Partitioning problem has been proved NP-hard [28] and two approximation algorithms have been proposed in [11, 28]. Intuitively, to achieve good solutions for the k-Edge-Partitioning problem, we need to partition traffic graph G into sub-graphs of at most k edges such that each sub-graph contains as few nodes as possible. One key observation is that given a fixed number of edges of G, a sub-graph induced by the edges more likely contains fewer nodes if there are fewer components in the sub-graph. This is the basic idea behind the algorithms in [11, 28]. The algorithm in [28] guarantees

that each sub-graph is connected, while every sub-graph might contain only $\lceil k/2 \rceil$ edges in the worst case. The algorithm in [11] does not guarantee that each sub-graph is connected, instead it guarantees that the total number of components over all sub-graphs is bounded above and each sub-graph contains exactly k edges.

In this section, we propose two linear time approximation algorithms for the k-Edge-Partitioning problem based on a novel graph partitioning approach. Both algorithms utilize the similar idea as that of the algorithm in [11], and they guarantee the total number of components over all sub-graphs is bounded above. Our algorithms use at most $\lceil (1 + \frac{1}{k})|E(G)| \rceil + \left\lfloor \frac{|V(G)|}{4} \right\rfloor$ SADMs and achieve a better upper bound than previous algorithms of [11] and [28] as long as $\Delta_{odd} > \frac{|V(G)|}{2}$ (i.e., the traffic graph has a small number of even degree nodes) and $\frac{|E(G)|}{|V(G)|} > \frac{k}{4}$ (i.e., the traffic graph is relatively dense), respectively. It can be shown that our upper bounds are very close to the lower bounds for some instances. In addition, it is worth pointing out that both of our algorithms use the minimum number $\left\lceil \frac{|E(G)|}{k} \right\rceil$ of wavelengths. We also conduct simulations to evaluate the performances of our algorithms, and the results show that our algorithms outperform the previous algorithms.

3.2.1 Skeleton and skeleton cover

To solve the k-Edge-Partitioning problem, our algorithms use a novel approach of partitioning traffic graph G into special sub-graphs called *skeletons*. A skeleton S of G is a connected sub-graph of G that consists of a *backbone* and a set of *branches*, where the backbone is a path of G, and each branch is an edge of G such that at least one end-node of the edge is in the backbone. We say a branch $\{u, v\}$ is *attached* to node u in the backbone if u is a node in the backbone (notice that a branch may be attached to two nodes in the backbone). The *length* of a skeleton S, denoted as $l(S)$, is the length of its backbone. A *skeleton cover* $\mathcal{S}$ of graph G is a set of skeletons $\{S_1, S_2 \ldots, S_j\}$ which form an edge partition of G (i.e., $\bigcup_{i=1}^{j} E(S_i) = E(G)$ and $E(S_p) \cap E(S_q) = \emptyset$ for $p \neq q$), where j is the size of the skeleton cover (i.e., $j = |\mathcal{S}|$). We call $f : V(G) \rightarrow \mathcal{S}$ a *characteristic function* of skeleton cover $\mathcal{S}$ if f maps each node $u \in V(G)$ to a unique skeleton $S_i \in \mathcal{S}$ for some S_i with $u \in V(S_i)$. For each node $u \in V(G)$ and $f(u) = S_i$, u is called a *characteristic node* of S_i. The following properties on skeletons and skeleton covers play key roles in our algorithms.

Proposition 2 *For any skeleton S and integer t with $0 < t < |E(S)|$, S can be partitioned into two skeletons S_1 and S_2, such that $|E(S_1)| = t$ and $|E(S_2)| = |E(S)| - t$.*

Proof: We prove this proposition by constructing S_1 and S_2 from S. Assume that the backbone of S is $u_1 - u_2 - \cdots - u_{l(S)+1}$, where u_1 and $u_{l(S)+1}$ are the two end-nodes of the backbone. Initially S_1 is empty (i.e., $|E(S_1)| = 0$). We start from $i = 1$ to check whether the number of branches attached to u_i is less than $t - |E(S_1)|$. If so, we remove all such branches and edge $\{u_i, u_{i+1}\}$ from S, add them into S_1, and repeat the same process for next node u_{i+1} in the backbone. Otherwise we remove $t - |E(S_1)|$ such branches from S, add them into S_1, and terminate with $S_2 = S \backslash S_1$. It is easy to see that eventually both S_1 and S_2 are skeletons with $|E(S_1)| = t$ and $|E(S_2)| = |E(S)| - t$. □

By Proposition 2, it is easy to transform a skeleton cover to a k-edge partition of G with exactly k edges in each sub-graph except the last one (note that some sub-graphs might be disconnected). Especially we have the following proposition for any skeleton cover.

Proposition 3 *Any skeleton cover $\mathcal{S} = \{S_1, S_2, \ldots, S_j\}$ of graph G can be transformed into a k-edge partition $\mathcal{E} = \{E_1, ..., E_W\}$ of G with $W = \left\lceil \frac{|E(G)|}{k} \right\rceil$, $|E_i| = k$ for $1 \leq i < W$, and $\sum_{E_i \in \mathcal{E}} |V_i| \leq \lceil (1 + \frac{1}{k})|E(G)| \rceil + (j - 1)$.*

Proof: Let s_i and t_i be the end-nodes of the backbone of each skeleton $S_i \in \mathcal{S}$. We can connect $S_1, ..., S_j$ into one skeleton S^* by adding $(j - 1)$ virtual edges $\{t_i, s_{i+1}\}$ for $1 \leq i \leq j-1$. According to Proposition 2, S^* can be cut into W skeletons $\{S_1^*, S_2^*, \ldots, S_W^*\}$, where $W = \left\lceil \frac{|E(G)|}{k} \right\rceil$, each skeleton S_i^* $(1 \leq i \leq W - 1)$ contains exactly k edges of G and j_i virtual edges, and S_W^* contains at most k edges of G and j_W virtual edges. We have $\sum_{i=1}^{W} j_i = j - 1$ since the total number of added virtual edges is $j - 1$. For a skeleton of $k + j_i$ edges, the maximum number of nodes in the skeleton is $k + j_i + 1$. Therefore the set of skeletons $\{S_1^*, S_2^*, \ldots, S_W^*\}$ becomes a k-edge-partition $\mathcal{E} = \{E_1, ..., E_W\}$ of G after deleting virtual edges, and

$$\sum_{E_i \in \mathcal{E}} |V_i| \leq |E(G)| + \sum_{i=1}^{W} j_i + W = \lceil (1 + \frac{1}{k})|E(G)| \rceil + (j - 1).$$

□

Proposition 4 *For any skeleton cover $\mathcal{S} = \{S_1, S_2, \ldots\}$ of graph G and integer t, if there exists a characteristic function f of $\mathcal{S}$ such that at most one skeleton in $\mathcal{S}$ contains less than t characteristic nodes, then $|\mathcal{S}| \leq \left\lceil \frac{|V(G)|}{t} \right\rceil$.*

Proof: Assume that at most one skeleton contains x (where $x < t$) characteristic nodes. By the definition of the characteristic function, each node $u \in V(G)$ is a characteristic

node for exactly one skeleton in $\mathcal{S}$. Since $|V(G)|$ is the total number of nodes, we have $x + (|\mathcal{S}| - 1)t \leq |V(G)|$, which implies that $|\mathcal{S}| \leq \lfloor (\frac{|V(G)|}{t} + \frac{t-1}{t}) \rfloor \leq \lceil \frac{|V(G)|}{t} \rceil$. □

By Proposition 4, if we can construct a skeleton cover $\mathcal{S}$ of G such that the number of characteristic nodes in each skeleton is bounded below, then the size of $\mathcal{S}$ is bounded above. After $\mathcal{S}$ is transformed to a k-edge partition of G, the total number of components over all sub-graphs in the k-edge partition will be bounded above.

In the following we propose two linear time approximation algorithms for the k-Edge-Partitioning problem based on constructing skeleton covers of the traffic graph. The first algorithm kEP constructs a skeleton cover with size at most $\left\lceil \frac{|V(G)|}{4} \right\rceil$ for traffic graph G. The second algorithm SpanT_Euler constructs a skeleton cover with the help of Euler path construction, and achieves an upper bound that is at least as good as the one obtained by Algorithm kEP. In addition, both Algorithm kEP and Algorithm SpanT_Euler use the minimum number $\left\lceil \frac{|E(G)|}{k} \right\rceil$ of wavelengths.

3.2.2 Algorithm kEP

The algorithm

We first give a brief review on the previous algorithms [11, 28] for the k-Edge-Partitioning problem, since our algorithm is designed in order to overcome the deficiencies of these algorithms. The algorithm in [11] first adds virtual edges between odd-degree nodes to make every node of G have even degree. Then an Euler path is constructed, and cut into segments such that each segment contains k edges of the original graph G and some virtual edges. A k-edge partition is obtained by removing virtual edges and merging duplicated copies of the same node in each segment. The algorithm in [28] transforms traffic graph G into an *open tree* as follows: it first computes a spanning tree T of G. Then for every edge $\{u, v\} \in E(G) \backslash E(T)$, renames node v as u_v and attaches edge $\{u, u_v\}$ to node u in T. It is clear that an open tree of G has the same edge set as G, and has duplicated copies for some nodes of G. Then the open tree is partitioned into subtrees such that $\lceil \frac{k}{2} \rceil \leq |E(T')| \leq k$ for each subtree T', and a k-edge partition can be obtained by merging duplicated copies of the same node in each subtree.

As we mentioned previously, in order to achieve good solutions for the k-Edge-Partitioning problem, the obtained k-edge partition should contain as few components as possible over all sub-graphs. For the algorithm in [11], the number of components depends on the number

of odd-degree nodes that can be as large as $|V(G)|$ in the worst case. In addition, the adjacent edges in G might be separated far away in the constructed Euler path, therefore each sub-graph G^* in the obtained k-edge partition could be sparse in term of the ratio $\frac{|E(G^*)|}{|V(G^*)|}$. Similar problem might occur for the algorithm in [28], since the adjacent edges in G might be separated far away in the constructed open tree.

We propose Algorithm kEP for the k-Edge-Partitioning problem. Our algorithm first constructs a skeleton cover $\mathcal{S}$ of traffic graph G. Then all the constructed skeletons are connected together by adding virtual edges to form a virtual skeleton, which covers all the edges of G exactly once. Then the virtual skeleton can be partitioned into segments, each of which contains exactly k edges of G and some virtual edges according to Proposition 2. We notice that duplicated copies of a node of G might exist in a segment. In order to get a k-edge partition of G, we need to merge duplicated copies of the same node as follows: for any two edges $\{v, w\}$ and $\{x, y\}$ in a segment, assume v and x are two duplicated copies of the same node in G (i.e., $\{v, w\}$ and $\{x, y\}$ are adjacent edges in G), we delete edge $\{x, y\}$ from the segment and add a new edge $\{v, y\}$ into the segment. Therefore, a k-edge partition can be obtained by removing virtual edges and merging duplicated copies of the same node in each segment. The pseudo code of Algorithm kEP is given in Figure 3.1.

Algorithm kEP(G,k)
Input: An undirected graph G and integer k.
Output: A k-edge partition of G.
begin
  Call Subroutine **FindSkeleton**(G) to construct a skeleton cover $\mathcal{S} = \{S_1, ..., S_s\}$;
  Let u_i and v_i be the end-nodes of the backbone of S_i;
  Connect $S_1, ..., S_s$ into one virtual skeleton S^* by adding $(s-1)$ virtual edges $\{v_i, u_{i+1}\}$ for $1 \leq i \leq s-1$;
  Cut S^* into $l = \lceil (|E(S^*)| - s + 1)/k \rceil = \lceil |E(G)|/k \rceil$ segments such that
    each of the first $l-1$ segments has k edges of G;
  Remove virtual edges and merge duplicated copies of the same node in each segment;
end.

Figure 3.1: Pseudo code of Algorithm kEP.

The key part of Algorithm kEP is Subroutine FindSkeleton, which constructs a skeleton cover $\mathcal{S}$. In order to avoid the deficiencies of previous algorithms [11, 28], $\mathcal{S}$ is constructed based on a rooted spanning tree T of traffic graph G (without loss of generality, we assume that the traffic graph is connected, since otherwise we can deal with each component of the traffic graph individually). Roughly speaking, the backbone of each skeleton is extracted

from $E(T)$, and edges of $E(G)\backslash E(T)$ are attached as branches (a few edges from $E(T)$ might also be attached as branches). Thus each skeleton is very likely to be a relatively dense sub-graph of G. In addition, we construct $\mathcal{S}$ in a way that each skeleton in $\mathcal{S}$ has at least four characteristic nodes (it can be easily show that there exist instances where four characteristic nodes is the maximum worst case guarantee we can obtain), which guarantees that the size of skeleton cover $\mathcal{S}$ is at most $\left\lceil \frac{|V(G)|}{4} \right\rceil$ for traffic graph G. Figure 3.2 gives an example on constructing a skeleton cover for a given traffic graph.

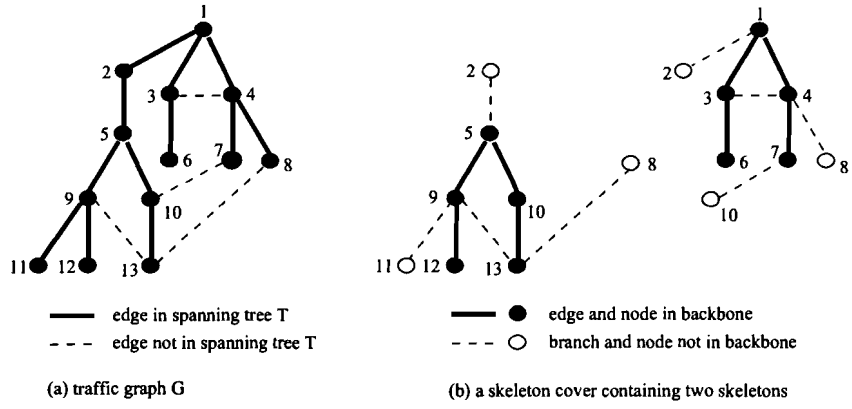

Figure 3.2: Illustration of constructing skeleton cover.

Now we give the details of Subroutine FindSkeleton. Let $\mathcal{S}$ be the set of skeletons extracted so far, and $F = E(G) \setminus (\cup_{S_i \in \mathcal{S}} E(S_i))$ be the set of edges of G that have not been included in the skeletons of $\mathcal{S}$. Let G_F be the graph induced by the edges in set F. Initially, $\mathcal{S}$ is an empty set and $G_F = G$. Let T be a rooted spanning tree of graph G_F. We call the edges in $E(T)$ *black edges* and edges in $E_w = F \setminus E(T)$ *white edges*. The skeleton extraction is done in iterations. For each iteration, a subtree T_u with height two and the largest depth in T is selected. Backbones are extracted from $E(T_u)$ and branches are extracted from E_w (a few branches might be from T_u as well) to form skeletons. The constructed skeletons are included into $\mathcal{S}$, and edges appearing in the skeletons are removed from $E(T)$ and E_w. The largest depth of T_u guarantees that T is connected after the removal of edges appearing in extracted skeletons, and T is always maintained to be a spanning tree of G_F. The skeleton extraction is repeated until $h(T) \leq 1$. Once $h(T) \leq 1$, the last skeleton is extracted if T is

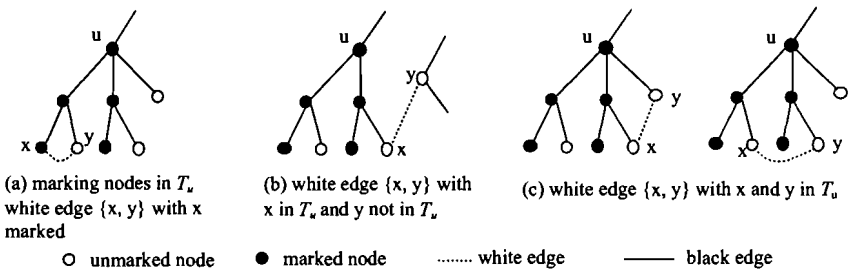

Figure 3.3: Marking nodes of T_u and white edge types.

not trivial.

Let T_u be a selected subtree of height two. T_u may only contain three nodes of G (i.e., T_u is a path $x - x_p - u$). In this case, we can not get a skeleton with at least four characteristic nodes from T_u. Instead, we contract the edge $\{x, x_p\}$ into a *compact node* x_c and go to the next iteration. If T_u contains at least four nodes of G or a compact node, skeletons are extracted. There are two major steps in the extraction. The first step is to mark the nodes to be included in the backbone of a skeleton and update the spanning tree T (and thus T_u) such that each white edge incident to a node of T_u is adjacent to at least one marked node. The second step extracts skeletons.

In the first major step, we mark every non-leaf node and every compact node in T_u. For T_u, we define

$$D_l(u) = \{x | x \text{ is a descendant of } u \text{ and } d_T(x, u) = l\}.$$

For every non-leaf node $v \in D_1(u)$, if there is no compact node in $D_1(v)$, then all nodes of $D_1(v)$ are unmarked and we arbitrarily mark one node in $D_1(v)$. Since each compact node has been marked (we will show later that each $D_1(v)$ contains at most one compact node), exactly one node in each $D_1(v)$ is marked (see Figure 3.3(a)).

After the marking, each white edge $\{x, y\}$ incident to a node of T_u belongs to one of the following types: (a) at least one of x and y is marked, (b) x is an unmarked node of T_u and y is a node of $T \setminus T_u$, and (c) x and y are unmarked nodes of T_u (see Figure 3.3 for examples of each type). We update T (and thus T_u) by exchanging some white edges of the types (b) and (c) with the edges in $E(T)$ (i.e., black edges), such that the following properties hold:

- **Property 1:** Each white edge $\{x, y\}$ incident to a node of T_u has either x or y marked;

- **Property 2:** T is always maintained as a spanning tree of G_F;
- **Property 3:** The height of T_u is not increased after the updating.

Property 1 guarantees that each white edge incident to a node of T can be included in a skeleton as a branch. Property 2 guarantees that no edge will be missed when extracting skeletons, since every edge of G_F is either in the spanning tree T (i.e., a black edge) or incident to a node in T (i.e., a white edge). Property 3 is used to make the second major step (i.e., the skeleton extraction step) less complicated, as we only need to consider a subtree of height two. Based on the three properties above, the key in extracting a skeleton is to find the backbone of the skeleton. From now on, when we say a skeleton S with a backbone $x_1 - x_2 - ... - x_s$ is extracted, we mean that S is extracted with $x_1 - x_2 - ... - x_s$ as the backbone, and all white edges incident to the nodes of the backbone as branches. Then S is included into $\mathcal{S}$ and the edges of $E(S)$ are deleted from G_F.

To satisfy Property 1, we need to process each white edge of the type (b) or (c), without violating Properties 2 and 3. Recall that the parent of a node $x \in V(T)$ is denoted by x_p. For a white edge $\{x, y\}$ of the type (b), we change $\{x, y\}$ from white to black and change $\{x_p, x\}$ from black to white (i.e., add $\{x, y\}$ to $E(T)$ and delete $\{x_p, x\}$ from $E(T)$) (see Figure 3.4(b)). For a white edge $\{x, y\}$ of the type (c), we further have the following two sub-types: $(c1)$ at least one node of $\{x, y\}$ is in $D_1(u)$ and $(c2)$ $x, y \in D_2(u)$. For a type $(c1)$ white edge $\{x, y\}$, assume that $y \in D_1(u)$. We change $\{x, y\}$ from white to black, change $\{x, x_p\}$ from black to white, and mark the nodes x, y (see Figure 3.4(c1) for an example). For a type $(c2)$ white edge $\{x, y\}$, we do not change the color of $\{x, y\}$ but extract a skeleton from T_u with the backbone containing $\{x, y\}$ as follows.

- **Update-Extract**

  Case 1: $x_p \neq y_p$. Let x' and y' be the marked nodes in $D_1(x_p)$ and $D_1(y_p)$, respectively. We extract a skeleton S with the backbone $x' - x_p - x - y - y_p - y'$ (see Figure 3.4(c2-1)).

  Case 2: $x_p = y_p$ and $|D_1(x_p)| = 3$. We extract a skeleton S with the backbone $x' - x_p - x - y - x_p - u$ (see Figure 3.4(c2-2)).

  Case 3: $x_p = y_p$ and $|D_1(x_p)| > 3$. In this case, $D_1(x_p)$ contains at least one node z other than x', x, y. We extract a skeleton S with the backbone $x' - x_p - x - y - x_p - z$ (see Figure 3.4(c2-3)). If x', x, y, z are the only nodes in $D_1(x_p)$, $D_1(x_p)$ will become

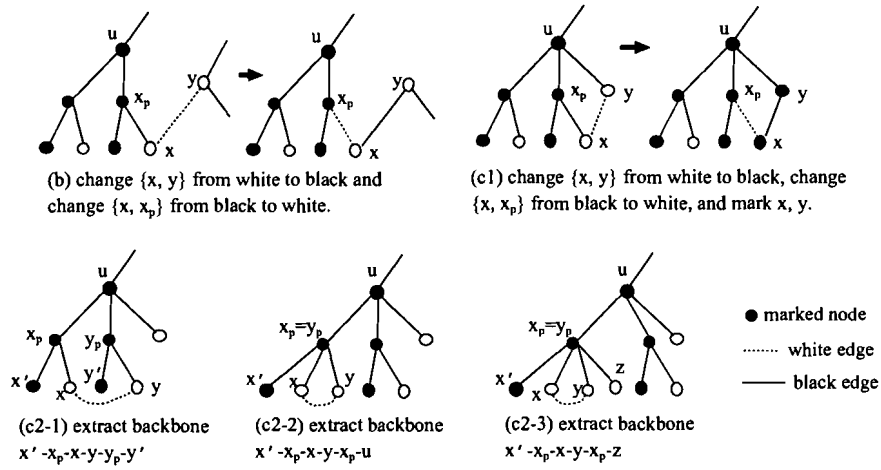

Figure 3.4: Updating T_u.

empty after the extraction of S, and we can attach the edge $\{x_p, u\}$ as a branch of S as well. Otherwise $D_1(x_p)$ is not empty and all nodes in $D_1(x_p)$ are unmarked, so we arbitrarily mark one node in $D_1(x_p)$.

It is easy to check that Properties 1, 2 and 3 are satisfied after the first major step. If the height of T_u is decreased and $T_u \neq T$, then we can go to the next iteration to select a new T_u of height two. Here we assume that after the marking step, either T_u has a height of two, or $T_u = T$ has a height of at most one. We execute the second major step to extract skeletons from T_u as follows.

- **Extract**

  Let M be the set of marked nodes in $D_2(u)$.

  Case 1: $|M| > 2$. We arbitrarily choose two nodes $x, y \in M$ and extract a skeleton S with the backbone $x - x_p - u - y_p - y$ (see Figure 3.5(a)). We repeat this process until $|M| \leq 2$.

  Case 2: $|M| = 2$. Let $M = \{x, y\}$. We extract a skeleton S with the backbone $x - x_p - u - y_p - y$. If the parent u_p of u exists, we extract the black edge $\{u, u_p\}$ as a branch of S as well.

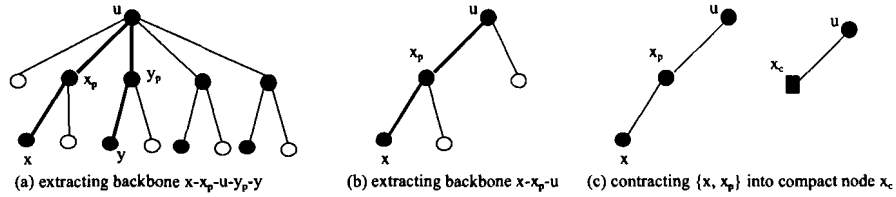

Figure 3.5: Extracting skeletons from T_u.

Case 3: $|M| = 1$. Let $M = \{x\}$. If $T_u = T$ (i.e., the skeleton to be extracted is the last one from T), $|V(T_u)| \geq 4$, or x is a compact node, we extract a skeleton S with the backbone $x - x_p - u$. (see Figure 3.5(b)). If the parent u_p of u exists, we extract $\{u, u_p\}$ as a branch of S as well.

Otherwise, T_u is a simple path $x - x_p - u$ and $T_u \neq T$ (see Figure 3.5(c)). To meet the constraint that each skeleton contains at least four characteristic nodes, we do not extract $x - x_p - u$ as a skeleton from T_u. However, T_u still has a height of two and may be selected in the next iteration. To break the deadlock, we contract the edge $\{x, x_p\}$ into a *compact node* x_c to reduce the height of T_u to one. It is observed that any node can not have more than one compact child, which guarantees that exactly one node in $D_1(v)$ will be marked in the first major step for each $v \in D_1(u)$ in the subtree T_u. After a compact node x_c is included in the backbone of a skeleton (recall that in the first major step we guarantee that a compact node will always be marked, and thus a compact node will always be in the backbone of a skeleton), we recover the edge $\{x, x_p\}$ from the node x_c and recover the edges incident to x or x_p accordingly.

Case 4: $|M| = 0$. In this case, $T_u = T$ and T_u has a height of at most one. If $|E(T)| \neq 0$, we extract the last skeleton S. It is observed that no white edge is incident to a marked child of u, since all of these white edges have been processed and included into some extracted skeletons. Therefore the single node u can be the backbone and the branches are $\{e | e \in E_w \cup E(T)\}$. We then extract the skeleton S.

The pseudo code of Subroutine FindSkeleton is given in Figure 3.6.

```
Subroutine FindSkeleton(G)
Input: Graph G.
Output: A skeleton cover S of G.
begin
   S := ∅; Compute a rooted spanning tree T of G;
   while h(T) > 1 do
   begin
      Find a subtree T_u with h(T_u) = 2 and the largest depth;
      /* Major step 1. */
      Mark every non-leaf node and every compact node of T_u;
      Mark one node of D_1(v) for every non-leaf node v ∈ D_1(u) if no compact node is in D_1(v);
      For each {x, y} ∈ E_w with x, y unmarked and at least one of x, y in V(T_u)
         switch {x, y} do
            case (x ∈ V(T_u) and y ∈ V(T)\V(T_u)): change white edge type (b);
            case (x ∈ D_1(u) ∪ D_2(u) and y ∈ D_1(u)): change white edge type (c1);
            case (x, y ∈ D_2(u) and x_p ≠ y_p): Update_Extract case 1;
            case (x, y ∈ D_2(u), x_p = y_p and |D_1(x_p)| = 3): Update_Extract case 2;
            case (x, y ∈ D_2(u), x_p = y_p and |D_1(x_p)| > 3): Update_Extract case 3;
      /* Major step 2. Let M be the set of marked nodes in D_2(u). */
      while |M| > 2 do
         { pick x, y ∈ M; Extract case 1; };
      switch M do
         case (M = {x, y}): Extract case 2;
         case (M = {x}): Extract case 3;
         case (M = ∅): Extract case 4;
   end of while
   Recover every compact node x_c in each S_i ∈ S;
end.
```

Figure 3.6: Subroutine for finding a skeleton cover $\mathcal{S}$ of G.

Analysis of the algorithm

In this section, we show that Algorithm kEP finds a k-edge partition that uses minimum number of wavelengths, and less SADMs than previous algorithms in most cases. First we prove the correctness of Subroutine FindSkeleton, that is, we prove that each edge of G appears in exactly one skeleton of $\mathcal{S}$.

Lemma 5 *Each edge of G appears in exactly one skeleton of $\mathcal{S}$.*

Proof: In major step 1 of Subroutine FindSkeleton, the nodes of T_u are marked in such a way that every edge incident to a node of T_u is incident to a node in the backbone of some skeleton $S \in \mathcal{S}$. This implies that every edge is included in some skeleton, either as an edge in the backbone or as an attached branch. In major step 2, once an edge is included in a

skeleton, it is removed from the graph. Thus, each edge appears in exactly one skeleton of $\mathcal{S}$. □

Next we show that the size of skeleton cover $\mathcal{S}$ is bounded above by $\left\lceil \frac{|V(G)|}{4} \right\rceil$.

Lemma 6 *The size of skeleton cover $\mathcal{S}$ is at most $\left\lceil \frac{|V(G)|}{4} \right\rceil$.*

Proof: Assume $\mathcal{S} = \{S_1, S_2, S_3, \ldots\}$, where the skeleton S_i is extracted before S_j for $i < j$. We define a characteristic function $f : V(G) \rightarrow \mathcal{S}$ as follows: for each $u \in V(G)$, $f(u) = S_i$ if $u \in V(S_i)$ and $u \notin V(S_j)$ for any $j > i$. We now show that for $i < j$, $|V(S_i) \setminus V(S_j)| \geq 4$, that is, each skeleton of $\mathcal{S}$ except the last one has at least four characteristic nodes. Each skeleton S_i is extracted in either **Update-Extract** or **Extract**.

- For Cases 1, 2, and 3 of **Update-Extract**, S_i has the backbone $x' - x_p - x - y - y_p - y'$, $x' - x_p - x - y - x_p - u$ and $x' - x_p - x - y - x_p - z$, respectively. The nodes $\{x', x, y, y'\}$, $\{x', x_p, x, y\}$, and $\{x', x, y, z\}$ do not appear in S_j $(i < j)$, respectively.
- For Cases 1 and 2 of **Extract**, S_i has the backbone $x - x_p - u - y_p - y$ and $x - x_p - u - y_p - y$, respectively. The nodes $\{x, x_p, y_p, y\}$ and $\{x, x_p, u, y_p, y\}$ do not appear in S_j $(i < j)$, respectively.
- For Case 3 of **Extract**, S_i has the backbone $x - x_p - u$. In this case, $|V(T_u)| \geq 4$ or x is a compact node if S_i is not the last extracted skeleton of $\mathcal{S}$. If x is a compact node, after recovering x, $|V(T_u)| \geq 4$. The nodes of $V(T_u)$ do not appear in S_j $(i < j)$ since S_i is the last skeleton extracted from T_u.
- For Case 4 of **Extract**, S_i is the last extracted skeleton of $\mathcal{S}$.

Therefore, we have $|V(S_i) \setminus V(S_j)| \geq 4$ for $i < j$, and the lemma holds, according to Proposition 4. □

Theorem 7 *Algorithm kEP finds a k-edge partition $\mathcal{E} = \{E_1, \ldots, E_W\}$ of G with $W = \left\lceil \frac{|E(G)|}{k} \right\rceil$, $|E_i| = k$ for $1 \leq i < W$, and $\sum_{E_i \in \mathcal{E}} |V_i| \leq \left\lceil (1 + \frac{1}{k})|E(G)| \right\rceil + \left\lfloor \frac{|V(G)|}{4} \right\rfloor$.*

Proof: The theorem can be easily proved based on Proposition 3 and Lemma 6. □

According to Theorem 7, Algorithm kEP uses at most $\left\lceil (1 + \frac{1}{k})|E(G)| \right\rceil + \left\lfloor \frac{|V(G)|}{4} \right\rfloor$ SADMs for a traffic graph G. In addition, Algorithm kEP uses the minimum number $\left\lceil \frac{|E(G)|}{k} \right\rceil$ of wavelengths since each sub-graph except the last one contains exactly k edges.

It is worth pointing out that there exist instances whose optimal solutions require as many as $(1+\frac{1}{k})|E(G)|+\frac{|E(G)|}{2k}$ SADMs, which is very close to the upper bound achieved by Algorithm kEP. Consider a traffic graph G that is a tree with height two and rooted at node r, and for each child u_i of r there exists exactly one child v_i of u_i, where $1 \leq i \leq \frac{|E(G)|}{2}$ (see Figure 3.7). Grooming factor k is an odd value, and for convenience we assume that $|E(G)|$ is a multiple of $2k$. It is easy to see that an optimal grooming scheme is to put k edges

$$\{r,u_i\},\{u_i,v_i\},\{r,u_{i+1}\},\{u_{i+1},v_{i+1}\},\ldots,\{r,u_{i+\lfloor\frac{k}{2}\rfloor-1}\},\{u_{i+\lfloor\frac{k}{2}\rfloor-1},v_{i+\lfloor\frac{k}{2}\rfloor-1}\},\{r,u_{i+\lfloor\frac{k}{2}\rfloor}\}$$

into a sub-graph, and put k edges

$$\{u_{i+\lfloor\frac{k}{2}\rfloor},v_{i+\lfloor\frac{k}{2}\rfloor}\},\{r,u_{i+\lfloor\frac{k}{2}\rfloor+1}\},\{u_{i+\lfloor\frac{k}{2}\rfloor+1},v_{i+\lfloor\frac{k}{2}\rfloor+1}\},\ldots,\{r,u_{i+k}\},\{u_{i+k},v_{i+k}\}$$

into a sub-graph, where $1 \leq i < \frac{|E(G)|}{2k}$. This optimal grooming scheme uses $(1+\frac{1}{k})|E(G)|+\frac{|E(G)|}{2k}$ SADMs, which is very close to our upper bound. In addition, our algorithm kEP does achieve the optimal solution for this specific instance.

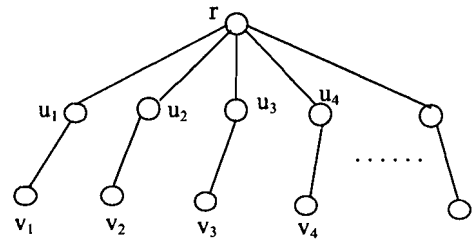

Figure 3.7: A tree with height two.

Constructing $\mathcal{S}$ can be done in $O(|E(G)|)$ time: First, it is well known that the spanning tree generation can be done in $O(|E(G)|)$ time. Second, each of the node marking and skeleton extraction takes $O(|E(G)|)$ time. This is because in each iteration, we process a subtree and the associated white edges (i.e., to mark the nodes and extract the skeletons from the subtree), and this subtree, together with the white edges, is deleted from the graph before proceeding to the next iteration. Third, after the skeleton cover is constructed, transforming the skeleton cover to a k-edge partition of G also takes $O(|E(G)|)$ time according

to Proposition 3. Therefore, Algorithm kEP runs in $O(|E(G)|)$ time, which is linear in the size of the input graph.

Goldschmidt *et al.* [28] established a lower bound by observing that each sub-graph in the best possible k-edge partition is a complete graph with exactly k edges. Therefore, the total number of nodes over all sub-graphs is bounded below by $\frac{|E(G)|}{k} \cdot \frac{1+\sqrt{8k+1}}{2}$. Based on this lower bound, we have the following theorem for Algorithm kEP.

Theorem 8 *For a graph G with m edges and n nodes, and an integer k, Algorithm kEP is a*

$$\frac{\frac{k}{m}(\lceil(1+\frac{1}{k})m\rceil + \lfloor\frac{n}{4}\rfloor)}{\frac{1+\sqrt{8k+1}}{2}}$$

approximation algorithm for the k-Edge-Partitioning problem.

Proof: Trivial. □

3.2.3 Algorithm SpanT_Euler

The k-Edge-Partitioning algorithms that we have discussed so far fall into two categories: spanning-tree based algorithms(the algorithm in [28] and Algorithm kEP) and Euler-path based algorithm (the algorithm in [11]). In this section, we propose a linear time approximation algorithm SpanT_Euler, which combines the spanning-tree based skeleton cover approach used by Algorithm kEP and the Euler-path based approach used in [11]. Algorithm SpanT_Euler tries to minimize the total number of components over all sub-graphs as well, and achieves an upper bound that is at least as good as the one obtained by Algorithm kEP. Algorithm SpanT_Euler also uses the minimum number $\left\lceil \frac{|E(G)|}{k} \right\rceil$ of wavelengths.

The algorithm

In order to achieve good solutions for the k-Edge-Partitioning problem, the obtained k-edge partition should contain as few components as possible over all sub-graphs. The algorithm in [11] constructs an Euler path of traffic graph G by adding virtual edges between odd-degree nodes. Then the Euler path is cut into segments and virtual edges are deleted to obtain a k-edge partition of G. However in the case that G contains a large number of odd-degree nodes, there will be a large number of virtual edges, whose deletion further implies a large number of components over sub-graphs in the k-edge partition. Algorithm

kEP constructs a skeleton cover of G, and then transform the skeleton cover into a k-edge partition. Skeleton are constructed using a spanning-tree based approach, which likely produces skeletons with small size, and thus the size of the skeleton cover is usually large. According to Proposition 3, a skeleton cover with large size yields a solution of the k-Edge-Partitioning problem with large value. In this section, we propose Algorithm SpanT_Euler that combines the techniques of constructing the Euler path and the skeleton cover. We intend to generate a skeleton cover of small size with the help of Euler path construction.

On constructing a skeleton cover, the following lemma holds.

Lemma 9 *If there exist l edge-disjoint paths which span every node in graph G, then G has a skeleton cover with size l.*

Proof: Each of the l paths can be considered as the backbone of a skeleton. Since the backbones span every node in graph G, other edges which do not appear in the backbones can be attached as branches. Thus the size of the constructed skeleton cover is exactly l. □

It is proved by Jaeger [36] that if the edge connectivity of G is at least four, then there exists a path that spans every node in G, which implies that there exists a skeleton cover with size one for graph G according to Lemma 9. Following a similar argument, we generalize the result of [36] as the following Lemma.

Lemma 10 *Let T be a spanning tree of G, and c be the number of components in graph $G(V(G), E(G)\backslash E(T))$, then G has a skeleton cover with size at most c.*

Proof: Let $V_{odd} \subseteq V(G)$ be the set of nodes having odd degree in graph $G(V(G), E(G)\backslash E(T))$. $|V_{odd}|$ must be even since the number of odd-degree nodes in any graph is even. Pair the nodes of V_{odd} arbitrarily and let $\mathcal{P} = \{p_1, p_2, \ldots, p_{\frac{|V_{odd}|}{2}}\}$ be the set of simple paths in T between each pair of nodes. For every edge $e \in E(T)$, let $\alpha(e)$ denote the number of paths (in $\mathcal{P}$) that contain edge e. Define

$$E_{odd} = \{e | e \in E(T) \text{ and } \alpha(e) \text{ is odd}\}, \text{ and}$$

$$E_{even} = \{e | e \in E(T), \alpha(e) \text{ is even, and } \alpha(e) \neq 0\}.$$

We prove in the following that V_{odd} is also the set of nodes that have odd degree in graph $G(V(G), E_{odd})$.

For every node $v \in V_{odd}$, let $E_v \subseteq E_{odd} \cup E_{even}$ be the set of edges adjacent to node v. We notice that v is an end-node for exactly one path in $\mathcal{P}$ (i.e., only one edge in the path is adjacent to v), and for any other path in $\mathcal{P}$, either v is a mid-node of the path (i.e., two edges in the path are adjacent to v) or v is not in the path (i.e., no edge in the path is adjacent to v). Therefore $\sum_{e \in E_v} \alpha(e)$ must be odd. We also know that

$$\sum_{e \in E_v} \alpha(e) = \sum_{e \in (E_v \cap E_{odd})} \alpha(e) + \sum_{e \in (E_v \cap E_{even})} \alpha(e),$$

so $|E_v \cap E_{odd}|$ must be odd, that is, every node $v \in V_{odd}$ has odd degree in graph $G(V(G), E_{odd})$. Similarly for every node $u \in V(G) \backslash V_{odd}$, since either u is a mid-node of some paths or u is not in any path, we have that $\sum_{e \in E_u} \alpha(e)$ must be even. That is,

$$\sum_{e \in E_u} \alpha(e) = \sum_{e \in (E_u \cap E_{odd})} \alpha(e) + \sum_{e \in (E_u \cap E_{even})} \alpha(e)$$

must be even, and thus $|E_u \cap E_{odd}|$ must be even. Therefore every node $u \in V(G) \backslash V_{odd}$ has even degree in graph $G(V(G), E_{odd})$.

Since V_{odd} is defined to be the set of nodes having odd degree in graph $G(V(G), E(G) \backslash E(T))$, every node in graph $G' = G(V(G), E_{odd} \cup (E(G) \backslash E(T)))$ has even degree. We know that $G(V(G), E(G) \backslash E(T))$ contains at most c components, so there are at most c components in graph G' as well. For each component in graph G', we can construct an Euler path since all nodes have even degree (for the component consisting of a single node, the Euler path is a special one consisting of the single node). It is clear that the Euler paths span every node in G. Thus we can construct a skeleton cover of G with size at most c according to Lemma 9. □

Notice that a component of graph G' contains either a single node or at least three nodes since each node has an even degree. A component is called *small* if it contains at most three nodes, otherwise called *large*. In the worst case, the size c of the skeleton cover constructed above could be as large as the number n of nodes in graph G (in this case, graph G is a tree T and removing the edges of T gives $c = n$ components, each of which is small component containing a single node). To further bound the size of the skeleton cover, we use Algorithm kEP to handle the small components of graph G'.

Given a graph G and a spanning tree T, let H be the sub-graph of G induced by the nodes of the small components of G'. For a component h of H, there is an edge $\{u, v\}$ and a large component g of G' such that $u \in V(h)$ and $v \in V(g)$, since graph G is connected. We

select one such g and edge $\{u, v\}$ for h. If h has at most three nodes, then we connect h to g via edge $\{u, v\}$ as an *attachment*. Otherwise, we use Algorithm kEP to extract skeletons from h with node u as the root of the spanning tree of h until at most three nodes are left in h, and let h' be the remaining sub-graph of h. We connect h' to g via edge $\{u, v\}$ as an attachment. Notice that h' is a path $u-x-y$ of three nodes and nodes x and y do not appear in any previously extracted skeletons (i.e., nodes x and y can be treated as characteristic nodes when they are included into some skeleton later). According to the above description, an attachment must be one of the following five cases: a single node, a 1-path, two 1-paths, a 2-path, or a triangle (see Figure 3.8(a)). We call an attachment *large* if it has at least two nodes, and *small* if it is a single node. Obviously, a large attachment contains at least two nodes which can be treated as characteristic nodes when they are included into some skeleton later. For each large attachment, we select a longest path from v to a node in the attachment as the *main-path* of the attachment (see Figure 3.8(a)).

For every large component g of G', we find an Euler cycle of g. For every node v in g, if there are at least two large attachments connected to v then we extract a skeleton with the backbone to be the main-paths of two large attachments. There are at least four characteristic nodes in the skeleton. We repeat this procedure until at most one large attachment is connected to v. After the process, we have an Euler cycle of g and each node v in the cycle is connected to at most one large attachment.

We view the Euler cycle of g as a ring with $|E(g)|$ edges (see Figure 3.8(b)). A node v of g may appear multiple times in the ring. We arbitrarily select one appearance as the real v and call any other a *copy* of v. All attachments connected to v are connected to the real v in the ring. We call node v in the ring a *special node* if it is connected to one large attachment. The extraction of skeletons for large component g are now divided into the following several cases:

- **Case 1:** there is no special node in the ring. A single skeleton is extracted with the backbone as an Euler path of g, and any small attachment connected to g can be treated as a branch (in the following, we will only mention the backbone of each skeleton, since once the backbone is obtained, it is easy to decide the corresponding branches). Notice that the backbone contains at least four nodes which do not appear in any other skeleton, and thus can be treated as characteristic nodes.

- **Case 2:** there is one special node v in the ring. A single skeleton is extracted with

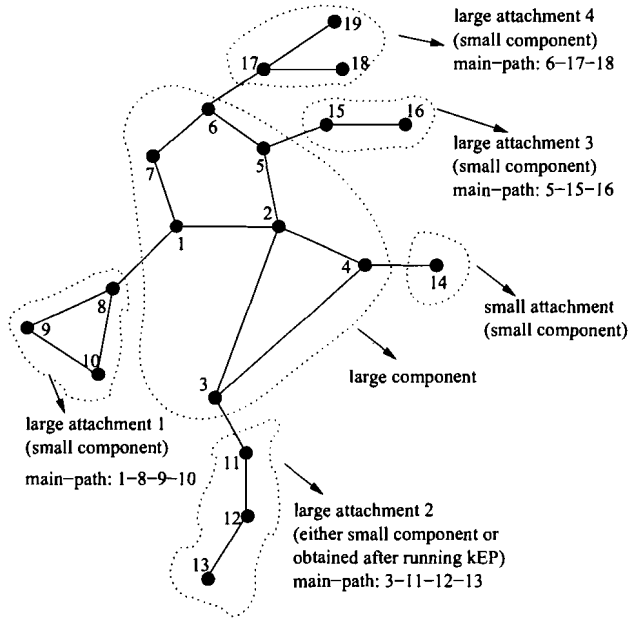

Figure 3.8: Skeleton extraction.

the backbone formed by an Euler path of g and the main-path of the large attachment connected to v. Similar as Case 1, this backbone also contains at least four characteristic nodes.

- **Case 3:** there are at least two special nodes in the ring. There will be two sub-cases for extracting the first skeleton:
  - Case 3.1: there are special nodes u and v such that the segment $[u, v]$ (including nodes u and v) in the ring does not contain any other special node and contains at least one other real node. We find such a $[u, v]$ and extract a skeleton with the backbone formed by $[u, v]$ and the main-paths of the two large attachments connected to u and v. If each real node in the backbone is treated as a characteristic node, then this backbone has at least seven characteristic nodes (see Figure 3.8(b) for $u = 1$ and $v = 3$).
  - Case 3.2: for any pair of special nodes u and v, if $[u, v]$ does not contain any other special node then $[u, v]$ does not contain any other real node. We find such a $[u, v]$ and extract a skeleton with the backbone formed by $[u, v]$ and the main-paths of the two large attachments connected to u and v. Notice that this backbone contains at least six characteristic nodes.

  After the first skeleton is extracted, let w be the first real node searched from v on the remaining segment of the ring and w' be the special node in the remaining segment such that $[w, w']$ does not contain any other special node. We extract a skeleton with the backbone formed by $[w, w']$ and the main-paths of the large attachments connected to w and w' (note that if w is not a special node, we only need to include one main-path. See Figure 3.8(b) for $w = 4$ and $w' = 5$). Notice that this backbone has at least four characteristic nodes. We repeat this procedure until all nodes in the ring are processed and included into some backbone. It is noticed that in the worst case the last backbone may contain only one real node as the characteristic node for Case 3.1, and contain only a main-path of some large attachment (i.e., there are three characteristic nodes in the backbone) for Case 3.2. Since the first backbone has at least seven characteristic nodes for Case 3.1 and six characteristic nodes for Case 3.2, there are at least four characteristic nodes in average in each skeleton.

After we process all large components of graph G' to extract skeletons, it is easy to verify

Algorithm SpanT_Euler
Input: An undirected graph G and integer k.
Output: A k-edge partition of G.
begin
Compute a spanning tree T of G;
Let c be the number of components in $G(V(G), E(G)\backslash E(T))$;
Let $V_{odd} \subseteq V(G)$ be the set of nodes having odd degree in $G(V(G), E(G)\backslash E(T))$;
Pair nodes of V_{odd}, and let $\mathcal{P} = \{p_1, p_2, \ldots, p_{|V_{odd}|/2}\}$ be simple paths in T between each pair of nodes;
Let $E_{odd} \subseteq E(T)$ be the set of edges appearing in odd number paths of $\mathcal{P}$;
Let $G' = G(V(G), E_{odd} \cup (E(G)\backslash E(T)))$;
Let H be sub-graph of G induced by the nodes of small components of graph G';
Call Algorithm kEP to extract skeletons from each component of H with at least four nodes;
Connect attachments to large components of G';
Process each large component to extract skeletons;
Form a skeleton cover consisting of all the constructed skeletons;
Transform skeleton cover to a k-edge partition of G;
end.

Figure 3.9: Pseudo code of Algorithm SpanT_Euler.

that we obtain a skeleton cover of graph G. The pseudo code of Algorithm SpanT_Euler is given in Figure 3.9.

Analysis of the algorithm

Lemma 11 *The size of the skeleton cover constructed in Algorithm SpanT_Euler is at most* $\min\left\{\left\lceil\frac{|V(G)|}{4}\right\rceil, c\right\}$.

Proof: Each skeleton is constructed to contain at least four characteristic nodes in average, therefore the size of the skeleton cover is obviously at most $\left\lceil\frac{|V(G)|}{4}\right\rceil$ according to Proposition 4. We will prove in the following that the size of the skeleton cover is bounded above by c as well.

The only possible reason, which might cause the size of the skeleton cover to exceed c, is that we may extract several skeletons from the same large component. However, it is noticed that each such skeleton (except the last one) contains at least one large attachment. For a large attachment, either it contains at least a small component of graph G', or it is obtained after we extract skeletons by calling Algorithm kEP. For the former case, it is easy

to conclude that cutting a large component into skeletons will not make the number of constructed skeletons exceed c, since each such skeleton contains at least one small component of G'. For the latter case, the large attachment is obtained after extracting skeletons from some component h of H (recall that H is the sub-graph of G induced by the nodes of the small components of G'). We also know that each such extracted skeleton contains at least four characteristic nodes, and h is composed of small components containing at most three nodes. Therefore, the number of extracted skeletons from h must be at least one less than the number of small components contained in h. Therefore, we can have the same conclusion as the former case. In summary, we conclude that the total number of constructed skeletons in Algorithm SpanT_Euler can not exceed c. □

Theorem 12 *Algorithm SpanT_Euler finds a k-edge partition $\mathcal{E} = \{E_1, ..., E_W\}$ of G with $W = \left\lceil \frac{|E(G)|}{k} \right\rceil$, $|E_i| = k$ for $1 \leq i < W$, and*
$\sum_{E_i \in \mathcal{E}} |V_i| \leq \left\lceil (1 + \frac{1}{k})|E(G)| \right\rceil + \min \left\{ \left\lfloor \frac{|V(G)|}{4} \right\rfloor, (c-1) \right\}$.

Proof: The theorem can be easily proved based on Proposition 3 and Lemma 11. □

By Theorem 12, at most $\left\lceil (1 + \frac{1}{k})|E(G)| \right\rceil + \min \left\{ \left\lfloor \frac{|V(G)|}{4} \right\rfloor, (c-1) \right\}$ SADMs are used by Algorithm SpanT_Euler. It is also noticed that the algorithm uses the minimum number $\left\lceil \frac{|E(G)|}{k} \right\rceil$ of wavelengths for a traffic graph G.

Similarly, we have the following theorem for Algorithm SpanT_Euler.

Theorem 13 *For a graph G with m edges and n nodes, and an integer k, Algorithm SpanT_Euler is a*

$$\frac{\frac{k}{m}(\lceil (1 + \frac{1}{k})m \rceil + \lfloor \frac{n}{4} \rfloor)}{\frac{1+\sqrt{8k+1}}{2}}$$

approximation algorithm for the k-Edge-Partitioning problem.

Proof: Trivial. □.

Since the spanning tree generation and Euler path construction can be done in $O(|E(G)|)$ time, it is clear that each step of Algorithm SpanT_Euler takes $O(|E(G)|)$ time. So Algorithm SpanT_Euler runs in $O(|E(G)|)$ time, which is linear in the size of the input graph.

3.3 Empirical results

In this section, we will conduct extensive simulations to evaluate the practical performances of our algorithms by comparing to existing algorithms. We implemented the algorithms

in [11, 28], our approximation algorithms kEP and SpanT_Euler to compare their performances on randomly generated traffic graphs. In generating graphs, we specify the number n of nodes and density ratio d that is used to calculate the number $m = n^{1+d}$ of edges. A graph of m edges and n nodes is generated by randomly connecting m pairs of nodes among all $n(n-1)/2$ possible pairs. We refer to the algorithms in [28] and [11] as Algo. 1 and Algo. 2 respectively. Figure 3.10 shows the empirical results for graphs of 36 nodes with different settings of density ratio d and grooming factor k. For each pair of values of d and k, we run the algorithms on 100 randomly generated traffic graphs and take the average number of required SADMs. We observe that the spanning-tree based Algo. 1 and Algo. kEP have better performances if density ratio d is smaller, and the Euler-path based Algo. 2 has better performance if density ratio d is larger. As we pointed out previously, Algorithm SpanT_Euler combines the techniques of constructing Euler path and skeleton cover. Therefore, Algorithm SpanT_Euler can be considered as a hybrid of the spanning-tree based approach and the Euler-path based approach. The empirical results in Figure 3.10 verify that Algorithm SpanT_Euler does take advantages of both approaches, and has better performance than all of the previous algorithms. Also, we would like to mention that for Algorithm SpanT_Euler and confidence interval $[1-5\%, 1+5\%]$, the confidence levels are 88%,93%,96%,98%,99% and 100% for density ratio $d = 0.1, 0.2, 0.3, 0.4, 0.5$ and 0.6 respectively. We ran simulations on graphs with different parameter settings as well, and the results present similar characteristics (see Figure 3.11 and Figure 3.12 for some other representative results).

3.4 Summary

In this chapter, we proposed two linear time approximation algorithms for the traffic grooming problem in the SONET/WDM UPSR network with unitary duplex traffic demands. Our algorithms achieve better upper bounds on the number of SADMs and experimental performances compared to previous ones, and they use the minimum number of wavelengths, which are also precious resources in optical networks. The gap between the upper bounds of our algorithms and the lower bound for arbitrary traffic graphs is still large (see Theorem 8 and Theorem 13). An interesting open problem is to further narrow the gap. The problem seems challenging for arbitrary graphs because the upper bounds of our algorithms have been shown very close to the optima for some sparse graph instances. The difficulty in

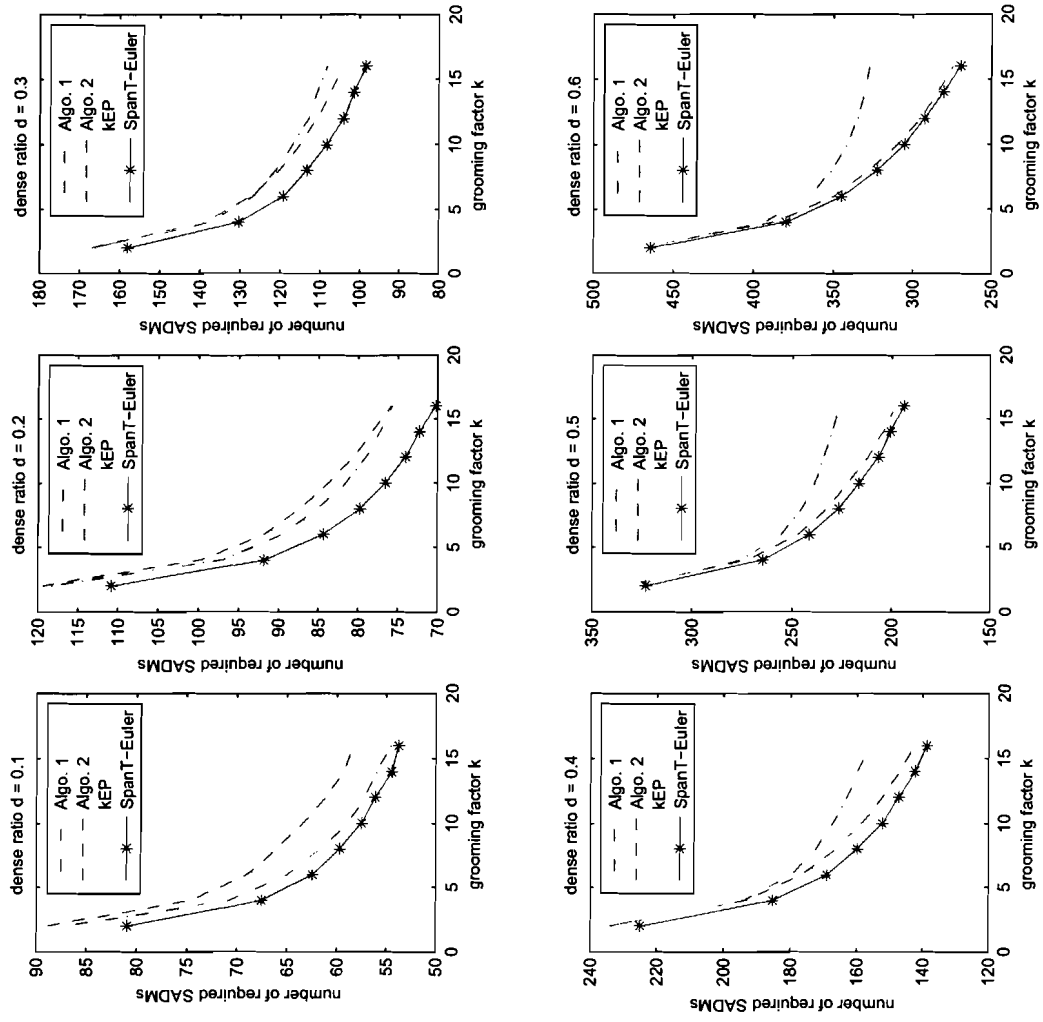

Figure 3.10: Empirical results on graphs with $n = 36$ nodes.

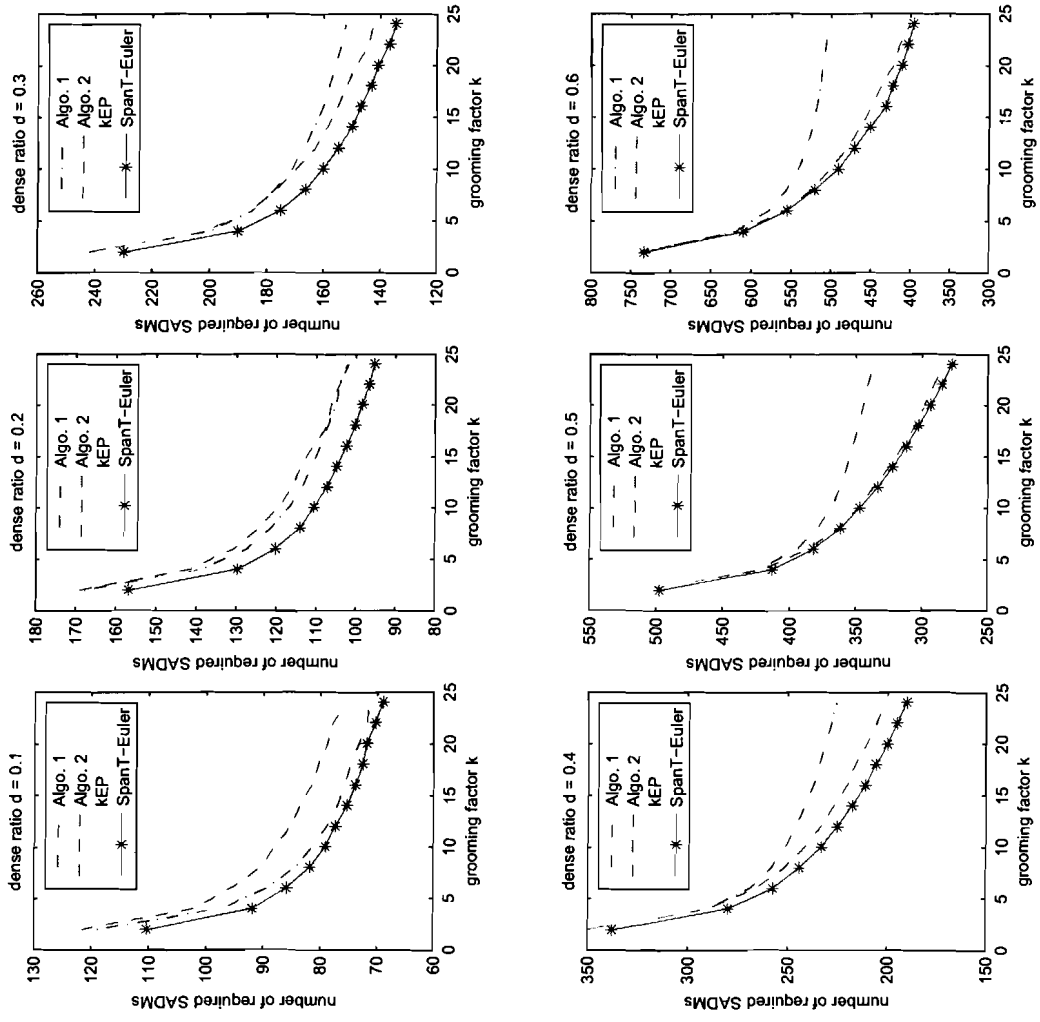

Figure 3.11: Empirical results on graphs with $n = 48$ nodes.

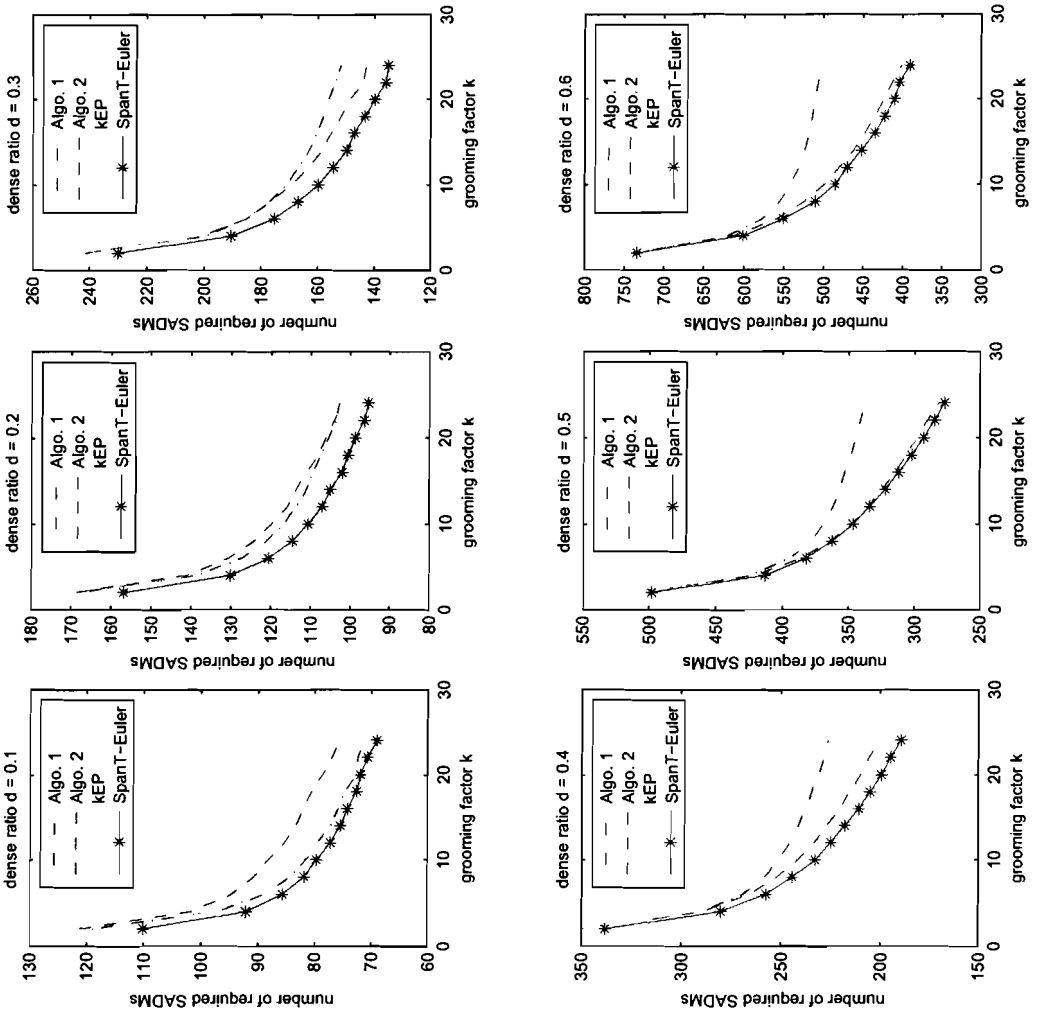

Figure 3.12: Empirical results on graphs with $n = 60$ nodes.

narrowing the gap is that sparse graphs (e.g., trees) are shown the barriers for improving the upper bounds, while the lower bound is based on the assumption that the graph can be partitioned into cliques, which implies the graph is either dense or has a special structure. One possible approach is to develop different lower bounds and algorithms for graphs with different densities. For dense graphs, it is interesting to design algorithms which partition the traffic graph into dense subgraphs, each of which is a clique or close to a clique. For sparse graphs, a better lower bound seems necessary.

Chapter 4

Traffic grooming in UPSR with regular traffic

In this chapter, we study the regular traffic pattern, which is considered as a generalization of the well-known all-to-all traffic pattern. We focus on the Unidirectional Path-Switched Ring (UPSR) networks as well. We prove that the traffic grooming problem is NP-hard for the regular traffic pattern in UPSR networks, and show that the problem does not admit a Fully Polynomial Time Approximation Scheme (FPTAS) unless $P = NP$. We further prove that the problem remains NP-hard even if the grooming factor is any value chosen from a subset of integers. We also propose a performance guaranteed algorithm to minimize the total number of required SADMs, and show that the algorithm achieves a better upper bound on the number of SADMs than previous algorithms applying on the regular traffic pattern. In addition, our algorithm always uses the minimum number of wavelengths, which are precious resources as well in optical networks.

4.1 Problem formulation

Recall that the traffic grooming problem in UPSR networks with duplex traffic pattern can be formulated as the k-Edge-Partitioning problem on the traffic graph. In this chapter, we study the regular traffic pattern, in which each network node is involved in r duplex traffic demand pairs in the network, where $1 \leq r \leq n-1$ for a network of n nodes. For the regular traffic pattern, the traffic graph is a regular graph, and the traffic grooming problem can be

formulated as the following k-Edge-Partitioning of Regular Graph problem:

k-Edge-Partitioning of Regular Graph Problem

Instance: A regular traffic graph $G(V, E)$ and integer $k \leq |E(G)|$.

Objective: Partition the edge set $E(G)$ into a collection of subsets $\mathcal{E} = \{E_1, E_2, \ldots, E_W\}$ (where $\bigcup_{i=1}^{W} E_i = E(G)$ and $E_p \cap E_q = \emptyset$ for $p \neq q$), such that $|E_i| \leq k$ for each $E_i \in \mathcal{E}$ and $\sum_{E_i \in \mathcal{E}} |V_i|$ is minimized, where V_i is the set of nodes in the sub-graph induced by edge set E_i.

The regular traffic pattern is a generalization of the well known all-to-all traffic pattern, for which the traffic grooming problem has been well studied in [7, 32, 42, 64]. The regular traffic pattern is of high practical interest, since optical network nodes usually have limited capacity on sourcing/ending traffic demands simultaneously at any time due to the hardware constraints. To the best of our knowledge, this is the first time that the regular traffic grooming problem is addressed. It has been shown that the k-Edge-Partitioning problem on arbitrary traffic graphs is NP-hard [28]. We prove that the k-Edge-Partitioning of Regular Graph problem remains NP-hard, and show that the problem does not admit a Fully Polynomial Time Approximation Scheme (FPTAS) unless $P = NP$. As well, we prove the problem remains NP-hard even if $k = a(a-1)/2$ for each integer $a \geq 3$. We also propose a performance guaranteed algorithm that achieves a better upper bound than previous algorithms designed for general duplex traffic patterns. In addition, our algorithm uses the minimum number of wavelengths, which are precious resources as well in SONET/WDM networks. We also conduct extensive simulations to evaluate the average performance of our algorithm.

4.2 Computational complexity

In this section we analyze the computational complexity of the k-Edge-Partitioning problem on regular graphs. We first prove that the k-Edge-Partitioning problem is NP-hard on r-regular graph for an arbitrary value of r, and further prove that the problem does not admit an FPTAS unless $P = NP$. In addition, we show that the problem remains NP-hard even if $k = a(a-1)/2$, where $a \geq 3$ is integer and $k < |E(G)|$.

4.2.1 NP-hardness

We consider the decision version of the k-Edge-Partitioning of Regular Graph (kEPRG) problem, and prove it is NP-complete. The kEPRG problem is stated as follows:

k-Edge-Partitioning of Regular Graph (kEPRG) Problem

Instance: An undirected regular graph $G(V, E)$, and integers k and T.

Question: Is there a partition of $E(G)$ into a collection of subsets $\mathcal{E} = \{E_1, E_2, \ldots, E_W\}$ (where $\bigcup_{i=1}^{W} E_i = E(G)$ and $E_p \cap E_q = \emptyset$ for $p \neq q$), such that $|E_i| \leq k$ for each $E_i \in \mathcal{E}$ and $\sum_{E_i \in \mathcal{E}} |V_i| \leq T$, where V_i is the set of nodes in the sub-graph induced by edge set E_i?

We give a two-step reduction from the Edge-Partition into Triangles (EPT) problem to the kEPRG problem. The EPT problem is known to be NP-complete [31] and stated as follows:

Edge-Partition into Triangles (EPT) Problem

Instance: An undirected graph $G(V, E)$ with $|E(G)| = m$.

Question: Is there a partition of $E(G)$ into sets $\{E_1, E_2, ..., E_{m/3}\}$ such that each E_i induces a triangle?

In the first step we show that the EPT problem on regular graphs is NP-complete by a reduction from the EPT problem.

Lemma 14 *The EPT problem remains NP-complete when the input graph G is regular.*

Proof: It is easy to see that if an odd-degree node exists in a graph, the graph can not be partitioned into triangles. Therefore, the EPT problem is NP-complete even if the input graph contains no odd-degree nodes. Otherwise, the EPT problem itself is not NP-complete. In what follows, we assume the input graph has no odd-degree nodes when we refer to an instance of the EPT problem.

The regular graph version of EPT is clearly in NP. Given an undirected graph G with maximum degree Δ (where Δ is even) , we construct a Δ-regular graph G^* as follows (for simplicity, we use (u, v, w) to denote a triangle with $\{u, v, w\}$ as node set):

1. For every node v in G with degree $\delta(v) < \Delta$, add triangles $\{(v, u_{v,i}, u_{v,i+1})\}$ for $i = 1, 3, 5, \ldots, \Delta - \delta(v) - 1$ to get graph G', where the $\Delta - \delta(v)$ nodes $u_{v,i}$'s and $u_{v,i+1}$'s are the new added nodes.

2. Make two extra copies of G', and assume the number of new added nodes in G' is q.

3. Re-label all the new added nodes as u_1, u_2, u_3, ..., u_{3q}.

4. If $3q < \frac{\Delta-2}{2}$

   (a) add $3p$ new nodes u_{3q+1}, u_{3q+2}, u_{3q+3}, ..., u_{3q+3p}, where p is the smallest integer satisfying $3q + 3p \geq \frac{\Delta-2}{2}$;

   (b) add p triangles $\{(u_{3q+i}, u_{3q+i+1}, u_{3q+i+2})\}$ for $i = 1, 4, 7, \ldots, 3p-2$;

   (c) set $q = q + p$.

5. Add $3q$ new nodes $w_1, w_2, w_3, \ldots, w_{3q}$ and q triangles $\{(w_i, w_{i+1}, w_{i+2})\}$ for $i = 1, 4, 7, \ldots, 3q-2$. Add another $3q$ new nodes $y_1, y_2, y_3, \ldots, y_{3q}$ and q triangles $\{(y_i, y_{i+1}, y_{i+2})\}$ for $i = 1, 4, 7, \ldots, 3q-2$.

6. Repeat the following for $i = 1, 2, 3, \ldots, \frac{\Delta-2}{2}$: add triangles $\{(u_j, w_{j\oplus i}, y_{j\oplus 2i})\}$ for $j = 1, 2, 3, \ldots, 3q$, where $\oplus$ is defined as mod $3q$ sum.

It is easy to verify that graph G^* constructed above is a Δ-regular graph. Figure 4.1 illustrates an example to construct regular graph G^* based on the given graph G (we make two copies for nodes y_1, y_2 and w_1 to avoid messy drawing).

Now we prove that G can be partitioned into triangles if and only if G^* can be partitioned into triangles. If G can be partitioned into triangles, according to the construction of G^*, G^* can be partitioned into triangles as well. If G^* can be partitioned into triangles, it is noticed that any triangle containing an edge from a copy of G can not contain any edge outside of that copy. So each copy of G by itself must be able to be partitioned into triangles. □

In the second step we show that the kEPRG problem is NP-complete by a reduction from the regular graph version of the EPT problem.

Theorem 15 *The kEPRG problem is NP-complete.*

Proof: The problem is clearly in NP. Given an instance of the regular graph version of EPT problem, where the input graph G is regular, we construct an instance of the kEPRG problem on the same graph with $T = m$ and $k = 3$, where $m = |E(G)|$.

If G can be partitioned into triangles, the triangle partition gives a solution for the kEPRG problem where $|E_i| = 3$ for each $E_i \in \mathcal{E}$ and $\sum_{E_i \in \mathcal{E}} |V_i| = m$.

We now prove that if G has a solution for the kEPRG problem with $k = 3$ and $T = m$, then it can be partitioned into triangles. It is noticed that $\sum_{E_i \in \mathcal{E}} |V_i| \geq m$ for $k = 3$, and the

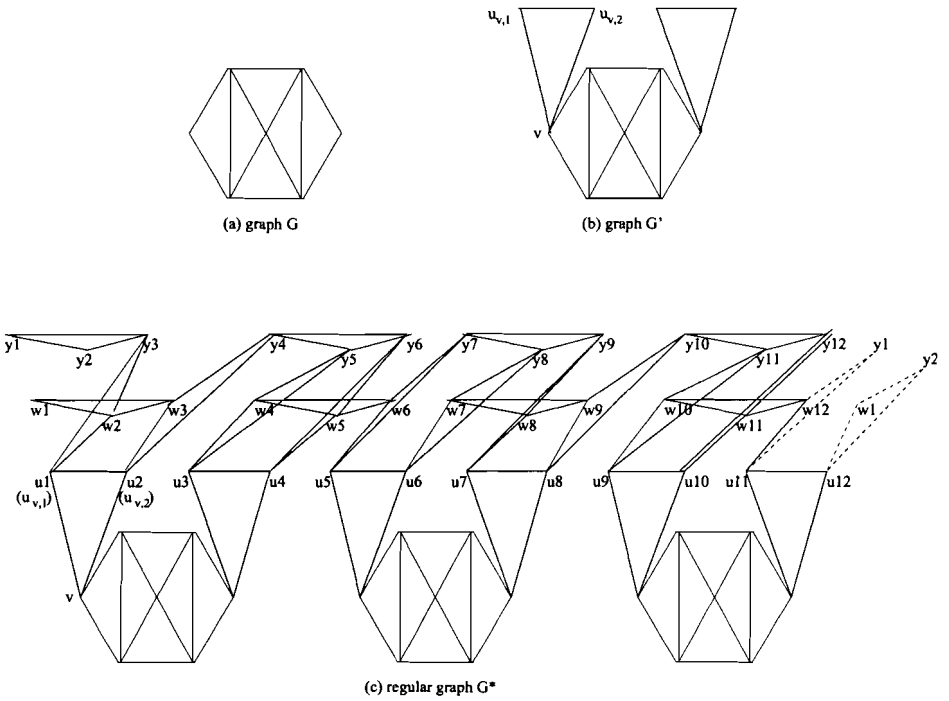

Figure 4.1: An example: graph G with $\Delta = 4$ and the corresponding 4-regular graph G^*.

equation holds if and only if G can be partitioned into complete graphs with 3 edges (i.e., triangles). Since we have $\sum_{E_i \in \mathcal{E}} |V_i| \leq T = m$, it must be the case that $\sum_{E_i \in \mathcal{E}} |V_i| = m$, which implies that G can be partitioned into triangles. □

4.2.2 The non-existence of FPTAS

We further prove that the kEPRG problem does not admit an FPTAS unless $P = NP$.

Theorem 16 *The kEPRG problem does not admit an FPTAS unless $P = NP$.*

Proof: Assume that kEPRG does admit an FPTAS. Now we construct an algorithm A that calls the FPTAS with parameter $\epsilon = \frac{1}{2m+1}$, where m is the number of edges of graph G. Since the time complexity of an FPTAS is polynomial in the problem size and also polynomial in $\frac{1}{\epsilon} = 2m + 1$, algorithm A is polynomial in the problem size (i.e., A is a

polynomial time algorithm for the kEPRG problem). Given any input instance I, let $A(I)$ denote the value of the solution by algorithm A and $OPT(I)$ denote the optimum. Since any solution to the problem must be integral value, we assume $A(I) = OPT(I) + i$ for some positive integer i. We have $OPT(I) + i = A(I) \leq (1 + \epsilon)OPT(I) = (1 + \frac{1}{2m+1})OPT(I)$, which implies that $OPT(I) \geq i(2m + 1)$. It is observed that $OPT(I) \leq 2m$ for any input graph with m edges, therefore $OPT(I) \geq i(2m + 1)$ holds only when $i = 0$, which implies that $OPT(I) = A(I)$. So A is an optimal algorithm with polynomial running time. But this can not be true for the NP-hard problem kEPRG unless $P = NP$. □

According to Theorem 16, it is worth pointing out that the following corollary holds for the k-Edge-Partitioning problem on arbitrary traffic graphs.

Corollary 17 *The k-Edge-Partitioning problem does not admit an FPTAS unless $P = NP$.*

4.2.3 NP-hardness for restricted k

In the above we proved that the kEPRG problem is NP-complete by showing the sub-problem of the kEPRG with $k = 3$ is NP-complete. In this section, we extend the NP-completeness result by showing that the kEPRG problem is NP-complete for every $k = a(a-1)/2$, where $a \geq 3$ is integer and $k < |E(G)|$.

Holyer proved in [31] that for each integer $a \geq 3$, it is NP-complete to determine whether an arbitrary graph can be edge-partitioned into sub-graphs isomorphic to the complete graph K_a. This NP-complete problem can be stated as the following Edge-Partition into K_a (EPKa) problem:

Edge-Partition into K_a (EPKa) Problem

Instance: An undirected graph $G(V, E)$ with $|E(G)| = m$, and an integer $a \geq 3$.

Question: Is there a partition of $E(G)$ into sets $\{E_1, E_2, ..., E_{\frac{m}{a(a-1)/2}}\}$ such that each E_i induces a complete graph K_a ?

We first show that for each integer $a \geq 3$, the EPKa problem on regular graphs is NP-complete by a reduction from the EPKa problem.

Lemma 18 *The EPKa problem remains NP-complete when the input graph G is regular.*

Proof: It is easy to see that if the degree of any node in a graph is not a multiple of $a - 1$, the graph can not be partitioned into K_a's. Therefore, the EPKa problem is NP-complete

even if the degree of every node in the input graph is a multiple of $a-1$. Otherwise, the EPKa problem itself is not NP-complete. In what follows, we assume that the degree of each node in the input graph is a multiple of $a-1$ when we refer to an instance of the EPKa problem.

The regular graph version of EPKa is clearly in NP. Given an undirected graph G with maximum degree Δ (where Δ is a multiple of $a-1$) , we construct a Δ-regular graph G^* as follows (where we use $(v_1, v_2, \ldots, v_a)$ to denote a complete graph K_a with $\{v_1, v_2, \ldots, v_a\}$ as node set):

1. For every node v in G with degree $\delta(v) < \Delta$, add K_a's $\{(v, u_{v,i}, u_{v,i+1}, \ldots, u_{v,i+(a-2)})\}$ for $i = 1, a, 2a-1, \ldots, \Delta - \delta(v) - (a-2)$ to get graph G', where the $\Delta - \delta(v)$ nodes $u_{v,i}$'s, $u_{v,i+1}$'s, ..., and $u_{v,i+(a-2)}$'s are the new added nodes.

2. Make $a-1$ extra copies of G', and assume the number of new added nodes in G' is q.

3. Re-label all the new added nodes in the a copies of G' as u_1^1, u_2^1, u_3^1, ..., u_{aq}^1.

4. If $aq < \frac{\Delta-(a-1)}{a-1}$

   (a) add ap new nodes u_{aq+1}^1, u_{aq+2}^1, ..., u_{aq+ap}^1, where p is the smallest integer satisfying $aq + ap \geq \frac{\Delta-(a-1)}{a-1}$;

   (b) add p K_a's $\{(u_{aq+i}^1, u_{aq+i+1}^1, \ldots, u_{aq+i+(a-1)}^1)\}$ for $i = 1, a+1, 2a+1, \ldots, ap-(a-1)$;

   (c) set $q = q + p$.

5. Repeat the following for $i = 2, 3, \ldots, a$: add aq new nodes u_1^i, u_2^i, u_3^i, ..., u_{aq}^i and q K_a's $\{(u_j^i, u_{j+1}^i, \ldots, u_{j+(a-1)}^i)\}$ for $j = 1, a+1, 2a+1, \ldots, aq-(a-1)$;

6. Repeat the following for $i = 1, 2, 3, \ldots, \frac{\Delta-(a-1)}{a-1}$: add K_a's $\{(u_j^1, u_{j\oplus i}^2, u_{j\oplus 2i}^3, \ldots, u_{j\oplus(a-1)i}^a)\}$ for $j = 1, 2, 3, \ldots, aq$, where $\oplus$ is defined as mod aq sum.

It is easy to verify that graph G^* constructed above is a Δ-regular graph.

Now we prove that G can be partitioned into K_a's if and only if G^* can be partitioned into K_a's. If G can be partitioned into K_a's, according to the construction of G^*, G^* can be partitioned into K_a's as well. If G^* can be partitioned into K_a's, it is noticed that any K_a containing an edge from a copy of G can not contain any edge outside of that copy. So each copy of G by itself must be able to be partitioned into K_a's. □

Now we show that the kEPRG problem is NP-complete for every $k = a(a-1)/2$, where $a \geq 3$ is integer and $k < |E(G)|$, by a reduction from the regular graph version of the EPKa problem.

Theorem 19 *For each integer $a \geq 3$, the kEPRG problem is NP-complete for $k = \frac{a(a-1)}{2}$, where $k < |E(G)|$.*

Proof: The problem is clearly in NP. Given an instance of the regular graph version of EPKa problem, where the input graph G is regular, we construct an instance of the kEPRG problem on the same graph with $L = \frac{2m}{a-1}$ and $k = a(a-1)/2$, where $m = |E(G)|$.

If G can be partitioned into K_a's, the K_a partition gives a solution for the kEPRG problem where $|E_i| = a(a-1)/2$ for each $E_i \in \mathcal{E}$ and $\sum_{E_i \in \mathcal{E}} |V_i| = \frac{2m}{a-1}$.

We now prove that if G has a solution for the kEPRG problem with $k = a(a-1)/2$ and $L = \frac{2m}{a-1}$, then it can be partitioned into K_a's. It is noticed that $\sum_{E_i \in \mathcal{E}} |V_i| \geq \frac{2m}{a-1}$ for $k = a(a-1)/2$, and the equation holds if and only if G can be partitioned into K_a's. Since we have $\sum_{E_i \in \mathcal{E}} |V_i| \leq L = \frac{2m}{a-1}$, it must be the case that $\sum_{E_i \in \mathcal{E}} |V_i| = \frac{2m}{a-1}$, which implies that G can be partitioned into K_a's. □

For $k = 2$ or $k = |E(G)|$, the kEPRG problem can be solved trivially. We have proved that the kEPRG problem is NP-complete for each $k = a(a-1)/2$, where $a \geq 3$ is integer and $k < |E(G)|$. It is interesting to derive the computational complexity of the kEPRG problem for other values of k with $3 < k < |E(G)|$. We conjecture that the kEPRG problem is NP-complete for every k with $3 \leq k < |E(G)|$, since it is difficult to get optimal solutions even for all-to-all traffic pattern (i.e., the traffic graph is a complete graph) for most values of k [7].

4.3 An approximation algorithm

In this section we propose Algorithm Regular_Euler with guaranteed performance for the kEPRG problem on r-regular graphs with $r \geq 2$. For r-regular graph G with $r = 0$ or 1, the traffic grooming is trivial: For $r = 0$, there is no edge in G and no grooming is needed; For $r = 1$, the edges of G constitute a matching of G. We partition $E(G)$ into subsets $E_1, E_2, \ldots, E_W$ (where $W = \left\lceil \frac{|E(G)|}{k} \right\rceil$) by putting arbitrarily k edges into each subset E_i, and this scheme uses $2|E(G)|$ SADMs that is optimal for 1-regular graph G.

For the k-Edge-Partitioning problem on arbitrary traffic graphs, several algorithms have been proposed in [11, 28]. Intuitively, to achieve good solutions for the k-Edge-Partitioning problem, we need to partition the traffic graph G into sub-graphs of at most k edges such that each sub-graph contains as few nodes as possible. One key observation is that given a fixed number of edges, a sub-graph with fewer components more likely contains fewer nodes. This is the basic idea behind the algorithms in [11, 28]. Our algorithm Regular_Euler utilizes a similar idea, and aims to minimize the total number of components over all sub-graphs. The algorithm uses at most $\lceil |E(G)|(1+\frac{1}{k}) \rceil$ SADMs for even $r \geq 2$, and $\lceil |E(G)|(1+\frac{1}{k}) \rceil + (\frac{3n}{2(r+1)} - 1)$ SADMs for odd $r \geq 3$, which are almost always better than previous algorithms applying on regular graphs.

First, we prove the following lemma about the lower bound on the size of the maximum matching of regular graphs. This lemma also gives a solution for one of the open problems proposed by Biedl *et al.* [10].

Lemma 20 *A r-regular graph G with $|V(G)| = n$ has a maximum matching containing at least $\frac{n}{2} \cdot \frac{r}{r+1}$ edges.*

Proof: It is proved by Vizing [49] that any simple graph G has an edge coloring with Δ or $\Delta + 1$ colors, where Δ is the maximum degree of G. The proof immediately yields an approximation algorithm to color any simple graph G with at most $\Delta + 1$ colors. Therefore, a r-regular graph G can be colored by $r + 1$ colors. Considering such an edge coloring, we notice that each set of edges with the same color is a matching of G. Since the total number of edges in G is $\frac{nr}{2}$, there must exist at least $\frac{nr/2}{r+1} = \frac{n}{2} \cdot \frac{r}{r+1}$ edges having the same color in the edge coloring. Therefore, $\frac{n}{2} \cdot \frac{r}{r+1}$ is a lower bound on the size of the maximum matching of r-regular graph. □

We prove the following lemma for r-regular graphs.

Lemma 21 *For r-regular graph G with $|V(G)| = n$ and $r \geq 2$, there exists a skeleton cover of size one if r is even, and there exists a skeleton cover of size at most $\frac{3n}{2(r+1)}$ if r is odd.*

Proof: For even value r (where $r \neq 0$), we can construct an Euler path of G. Such an Euler path is a skeleton of G without branches, therefore we obtain a skeleton cover of size one.

For odd value r (where $r \neq 1$), we first compute a maximum matching M of G. Then after deleting the edges in M from graph G, all the nodes saturated by M have degree $r-1$, all the unsaturated nodes have degree r, and there might be more than one components in

graph $G(V(G), E(G)\backslash M)$. We call a component in $G(V(G), E(G)\backslash M)$ an *even component* if every node in the component has degree $r-1$, and an *odd component* if there exists a node with degree r. Assume there are s even components and t odd components. Let $C_{even} = \{C'_1, C'_2, \ldots, C'_s\}$ be the set of even components, and $C_{odd} = \{C_1, C_2, \ldots, C_t\}$ be the set of odd components. It is noticed that there are $(n - 2|M|)$ odd-degree (i.e., the degree is r) nodes in the odd components, and every odd component C_i $(1 \leq i \leq t)$ contains at least two nodes $v_{i,1}, v_{i,2}$ with degree r since the number of odd-degree nodes in any graph must be even. We add $(n - 2|M|)/2 - 1$ virtual edges to change $(n - 2|M|) - 2$ odd-degree nodes to even-degree nodes (i.e., nodes with degree $r+1$) and connect the t odd components together into a connected graph G_{odd} at the same time. More specifically, we first add $t-1$ virtual edges $(v_{1,1}, v_{2,1}), (v_{2,2}, v_{3,1}), \ldots, (v_{t-2,2}, v_{t-1,1}), (v_{t-1,2}, v_{t,1})$ to connect the connect the odd components into one connected graph. Then for every pair of remaining odd-degree nodes except $v_{1,2}$ and $v_{t,2}$, we add a virtual edge between them. It is obvious that the new generated graph G_{odd} is connected and has exactly two odd-degree nodes $v_{1,2}$ and $v_{t,2}$. Therefore, an Euler path of G_{odd} can be found. For every even component, we can find an Euler path as well since each node of the component has even degree $r-1$. Taking the $s+1$ Euler paths as the backbones, we can construct a skeleton cover of size $s+1$. After deleting the $(n - 2|M|)/2 - 1$ virtual edges, we obtain a skeleton cover of G with size $(s+1) + (\frac{n-2|M|}{2} - 1) = s + \frac{n-2|M|}{2}$.

Since each node in an even component has degree $r-1$, each even component must contain at least r nodes. In graph $G(V(G), E(G)\backslash M)$, the total number of nodes with degree $r-1$ is $2|M|$, so the number of even components in graph $G(V(G), E(G)\backslash M)$ is at most $\frac{2|M|}{r}$, that is, $s \leq \frac{2|M|}{r}$.

We have $|M| \geq \frac{n}{2} \cdot \frac{r}{r+1}$ according to Lemma 20, so the size of the skeleton cover for odd value r is

$$s + \frac{n - 2|M|}{2} \leq \frac{2|M|}{r} + \frac{n - 2|M|}{2} = \frac{(4-2r)|M| + nr}{2r} \leq \frac{(4-2r) \cdot \frac{n}{2} \cdot \frac{r}{r+1} + nr}{2r} = \frac{3n}{2(r+1)}.$$

□

The pseudo code of Algorithm Regular_Euler is given in Figure 4.2.

Theorem 22 *Algorithm Regular_Euler finds a k-edge partition $\mathcal{E} = \{E_1, ..., E_W\}$ of r-regular graph G with $W = \left\lceil \frac{|E(G)|}{k} \right\rceil$, $|E_i| = k$ for $1 \leq i < W$, $\sum_{E_i \in \mathcal{E}} |V_i| \leq \lceil |E(G)|(1 + \frac{1}{k}) \rceil$ for even $r \geq 2$, and $\sum_{E_i \in \mathcal{E}} |V_i| \leq \lceil |E(G)|(1 + \frac{1}{k}) \rceil + (\frac{3n}{2(r+1)} - 1)$ for odd $r \geq 3$.*

Algorithm Regular_Euler
Input: An undirected r-regular graph G and integer k.
Output: A k-edge partition of G.
begin
  If r is even **then begin**
    Construct an Euler path of G;
    Cut Euler path into $W = \lceil |E(G)|/k \rceil$ segments such that each of first $W-1$ segments has k edges of G;
  end
  else begin
    Compute a maximum matching M of G;
    Connect odd components in $G(V(G), E(G) \backslash M)$ by virtual edges to form connected graph G_{odd};
    Construct an Euler path for G_{odd};
    Construct an Euler path for each even component in graph $G(V(G), E(G) \backslash M)$;
    Attach edges in M to Euler paths as branches and delete virtual edges to obtain skeleton cover $\mathcal{S} = \{S_1, ..., S_{\frac{3n}{2(r+1)}}\}$;
    Transform skeleton cover to a k-edge partition of G;
  end;
end.

Figure 4.2: Pseudo code of Algorithm Regular_Euler.

Proof: The theorem can be easily proved based on Proposition 3 and Lemma 21. □

If we apply the algorithms in [11] on r-regular graph, the upper bound on the number of SADMs is $\lceil |E(G)|(1+\frac{1}{k}) \rceil$ for even value r, and $\lceil |E(G)|(1+\frac{1}{k}) \rceil + \frac{n}{2}$ for odd value r. Therefore our algorithm Regular_Euler always gives a better upper bound. The algorithm in [28] applying on r-regular graph gives an upper bound of $\lceil |E(G)|(1+\frac{2}{k}) \rceil$. So for even value r, Algorithm Regular_Euler is always better. And for odd value r, Algorithm Regular_Euler is better when $\frac{|E(G)|}{k} > \frac{3n}{2(r+1)} - 1$ (the inequality holds as long as $r > \sqrt{3k}$). Since the grooming factor k is usually a constant, Algorithm Regular_Euler has a better performance than that of the algorithm in [28] when r is greater than some constant. It is also noticed that the algorithm in [28] may use as twice as the minimum number of wavelengths in the worst case. Algorithm kEP and SpanT_Euler applying on r-regular graph give an upper bound of $\lceil |E(G)|(1+\frac{1}{k}) \rceil + \lfloor \frac{n}{4} \rfloor$, therefore Algorithm Regular_Euler is always better except the only case of $r = 3$. In summary, Algorithm Regular_Euler almost always achieves a better upper bound than previous algorithms. In addition, it is worth pointing out that $W = \left\lceil \frac{|E(G)|}{k} \right\rceil$ corresponds to the number of wavelengths used by Algorithm Regular_Euler, and it is the minimum number of wavelengths required for a traffic graph G.

A maximum matching can be computed in $O(|V|^{\frac{1}{2}}|E(G)|)$ time, the Euler path construction can be done in $O(|E(G)|)$, and each of other steps in Algorithm Regular_Euler can be done in $O(|E(G)|)$ as well, therefore Algorithm Regular_Euler runs in $O(|V|^{\frac{1}{2}}|E(G)|)$ time.

4.4 Empirical results

In this section, we will conduct extensive simulations to evaluate the practical performance of our algorithm by comparing to existing algorithms. We implemented the algorithms in [11, 28], Algorithm kEP, Algorithm SpanT_Euler and Algorithm Regular_Euler to compare their performances on regular traffic graphs randomly generated by the regular graph generator from [2]. Figure 4.3 shows the empirical results for graphs of 36 nodes with different settings of degree r and grooming factor k, where for each value of r and k we collect the results from 100 randomly generated regular graphs and compute the average value. For Algorithm Regular_Euler, the confidence level for confidence interval [1-5%,1+5%] is about 94%. We refer to the algorithm in [28] and [11] as Algo. 1 and Algo. 2 respectively. The empirical results in Figure 4.3 verify that Algorithm Regular_Euler outperforms most of other algorithms (it is also observed that the performance of Algo. 2 is close to that of Algorithm Regular_Euler in most cases, which is due to the fact that both algorithms are Euler-path based algorithms. However, as we showed in the previous section, the worst case performance of Algorithm Regular_Euler is always better than that of Algo. 2). We ran simulations on graphs with different parameter settings as well, and the results present similar characteristics (see Figure 4.4 and Figure 4.5 for some representative results).

In Section 4.3 we showed that Algorithm Regular_Euler achieves a provable upper bound for the kEPRG problem, where the upper bound measures the worst case performance of the algorithm. We compare this worst case upper bound to the average performance of Algorithm Regular_Euler in Figure 4.6. We also include a lower bound for the kEPRG problem in the Figure. The lower bound is derived based on a lower bound for the k-Edge-Partitioning problem given in [28]. Goldschmidt *et al.* [28] established the lower bound for k-Edge-Partitioning problem by observing that each sub-graph in the best possible k-edge partition is a complete graph with exactly k edges. Therefore, the total number of nodes over all sub-graphs is bounded below by $\frac{|E(G)|}{k} \cdot \frac{1+\sqrt{8k+1}}{2}$. Given an integer a such that

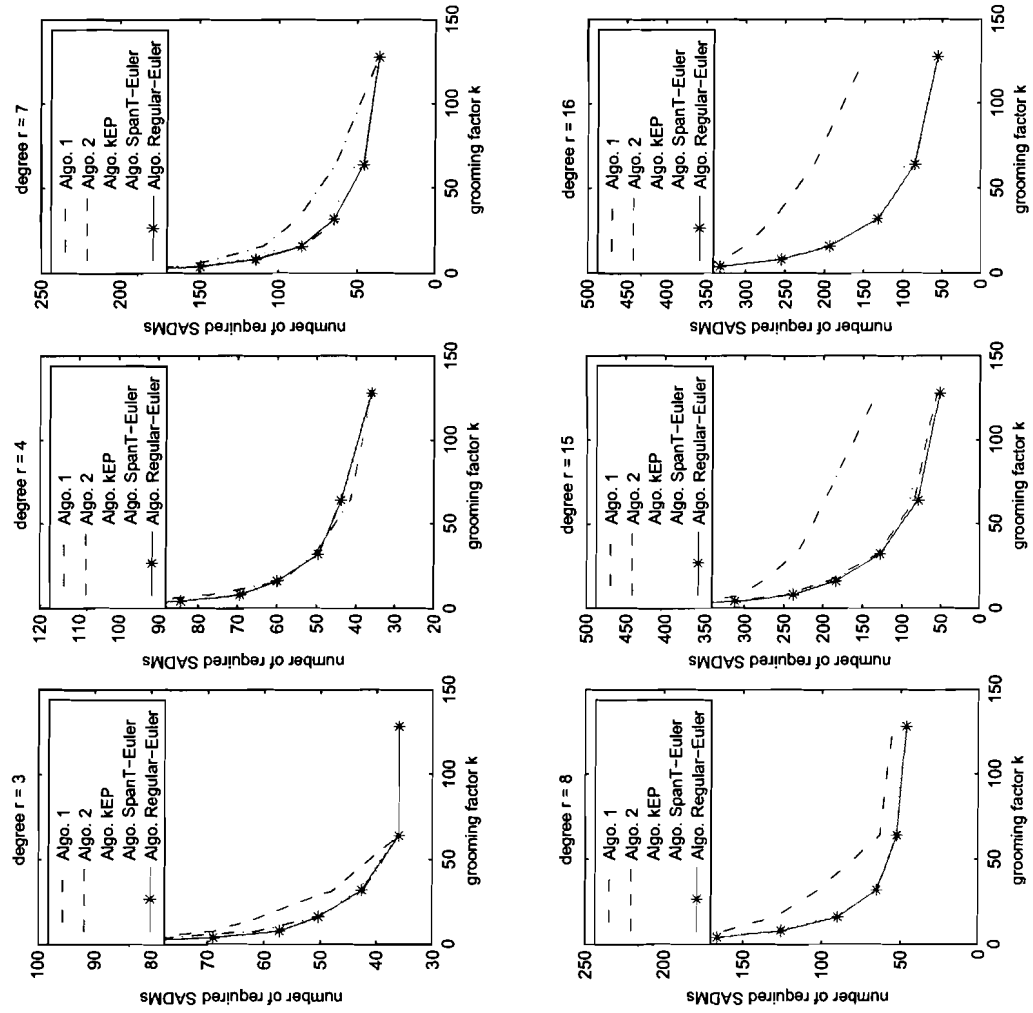

Figure 4.3: Algorithm Regular_Euler: empirical results on graphs with $n = 36$ nodes.

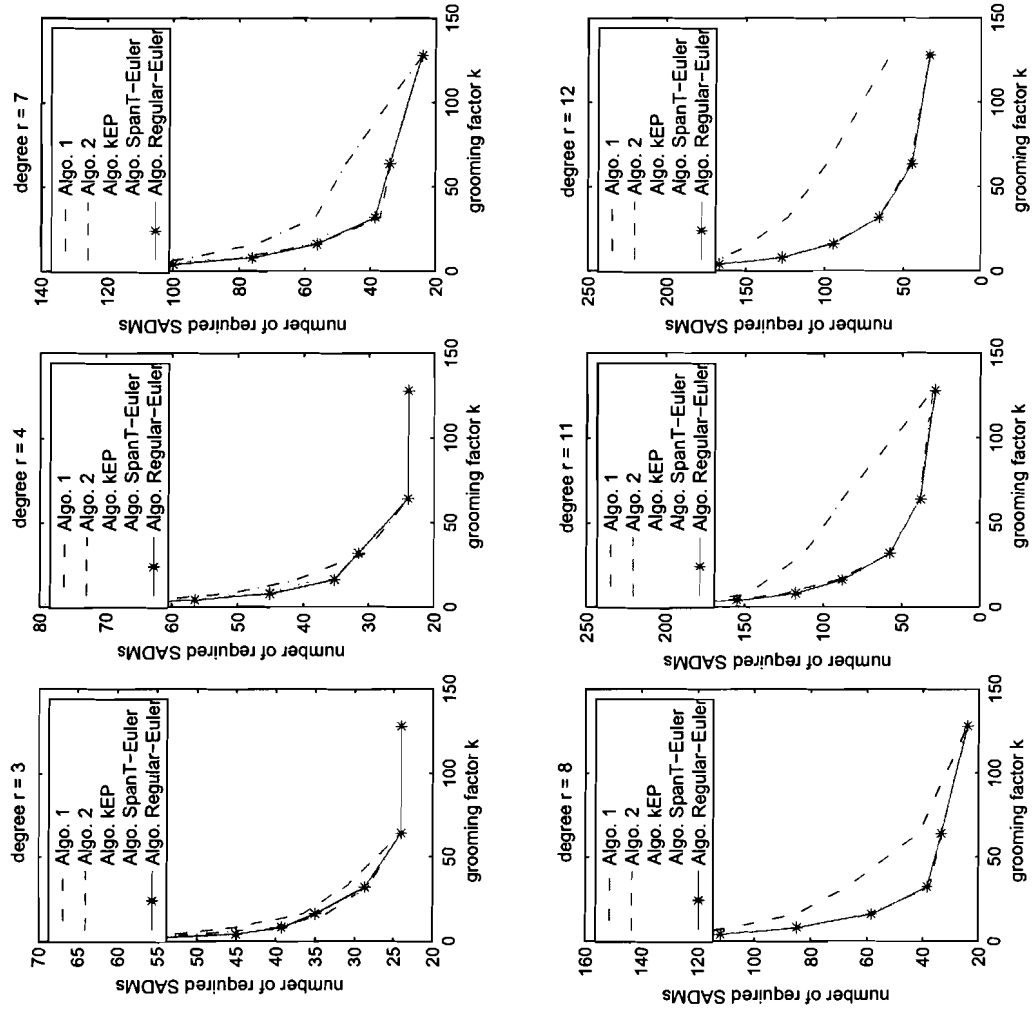

Figure 4.4: Algorithm Regular_Euler: empirical results on graphs with $n = 24$ nodes.

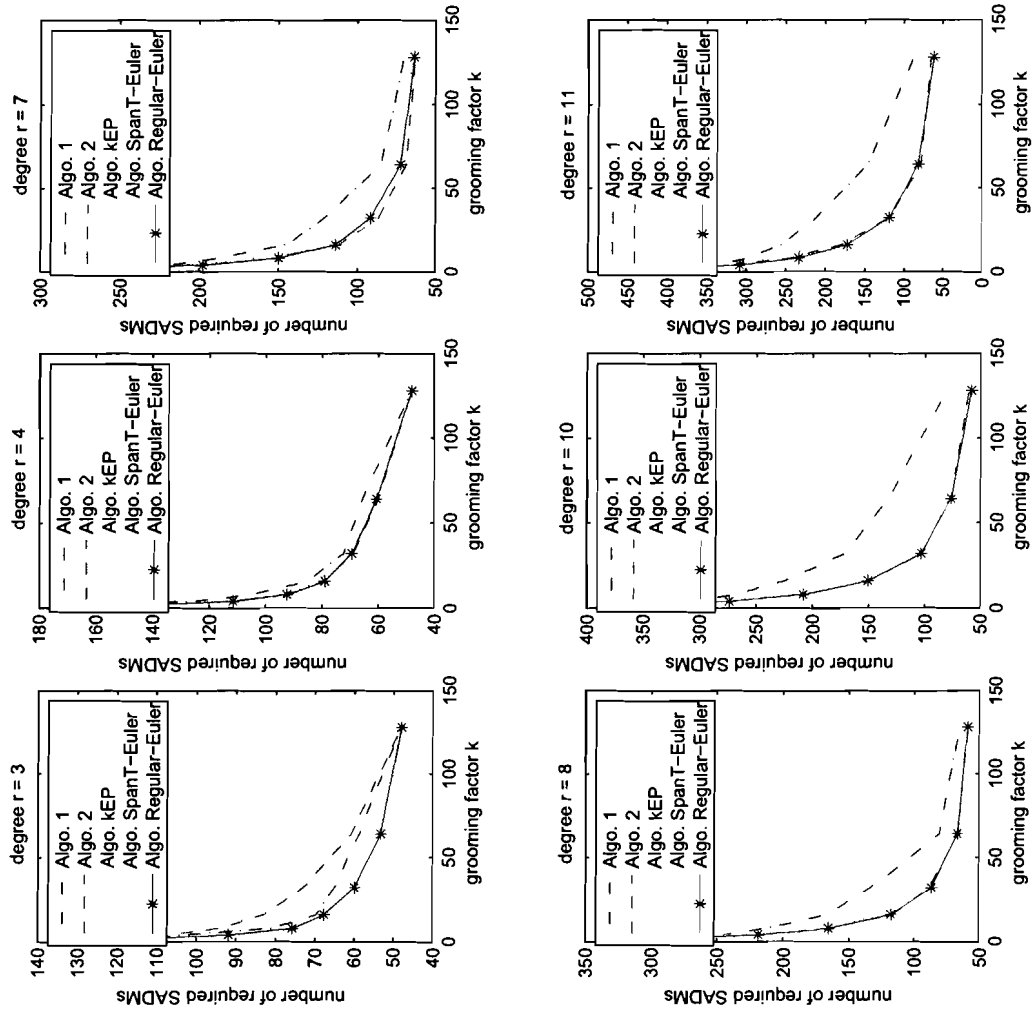

Figure 4.5: Algorithm Regular_Euler: empirical results on graphs with $n = 48$ nodes.

$a(a-1)/2 = k$, the best possible solution for the kEPRG problem is to partition the r-regular traffic graph into K_a's when $a \leq r$, and to partition the r-regular traffic graph into r-regular graphs with exactly k edges when $a > r$. Thus the lower bound for the kEPRG problem with a r-regular input graph can be expressed as

$$\text{Lower Bound} = \begin{cases} \frac{|E(G)|}{k} \cdot \frac{1+\sqrt{8k+1}}{2} & \text{if } \frac{1+\sqrt{8k+1}}{2} \leq r; \\ \frac{2|E(G)|}{r} & \text{if } \frac{1+\sqrt{8k+1}}{2} > r. \end{cases}$$

The empirical results demonstrate that although the gap between the provable upper bound and the lower bound is large, the average performance of Algorithm Regular_Euler is much closer to the lower bound. Simulations on graphs with different parameter settings present similar characteristics as well, and some other representative results are shown in Figure 4.7 and Figure 4.8.

4.5 Summary

In this chapter, we studied the traffic grooming problem on the SONET/WDM UPSR network with regular traffic pattern. We analyzed the computational complexity of the problem, and gave an algorithm with guaranteed performance. The algorithm achieves a better upper bound in most cases than previous algorithms and uses the minimum number of wavelengths as well. As proved in this chapter, the k-Edge-Partitioning of Regular Graph problem is NP-hard for any $k = a(a-1)/2$, where $a \geq 3$ is an integer. One interesting open problem is whether the problem remains NP-hard for any other value of k with $3 < k < E(G)$. When $k = a(a-1)/2$ for integer $a \geq 3$, we know the best solution to partition the traffic graph is that each sub-graph is a clique with a nodes and such a partitioning problem on arbitrary graphs is NP-hard, which were used to prove the NP-hardness of the problem. While for other values of k, the difficulty lies in that the sub-graphs with k edges in the best solution might not be unique, and the NP-hardness results of partitioning an arbitrary graph into sub-graphs containing k edges are not known as well. Another possible research direction is to derive tighter theoretical analysis to prove a better guaranteed performance for Algorithm Regular_Euler. As indicated by the empirical results, the average performance is always much better than the provable worst case performance. There might exists some room to improve the approximation ratio if a better lower bound can be derived.

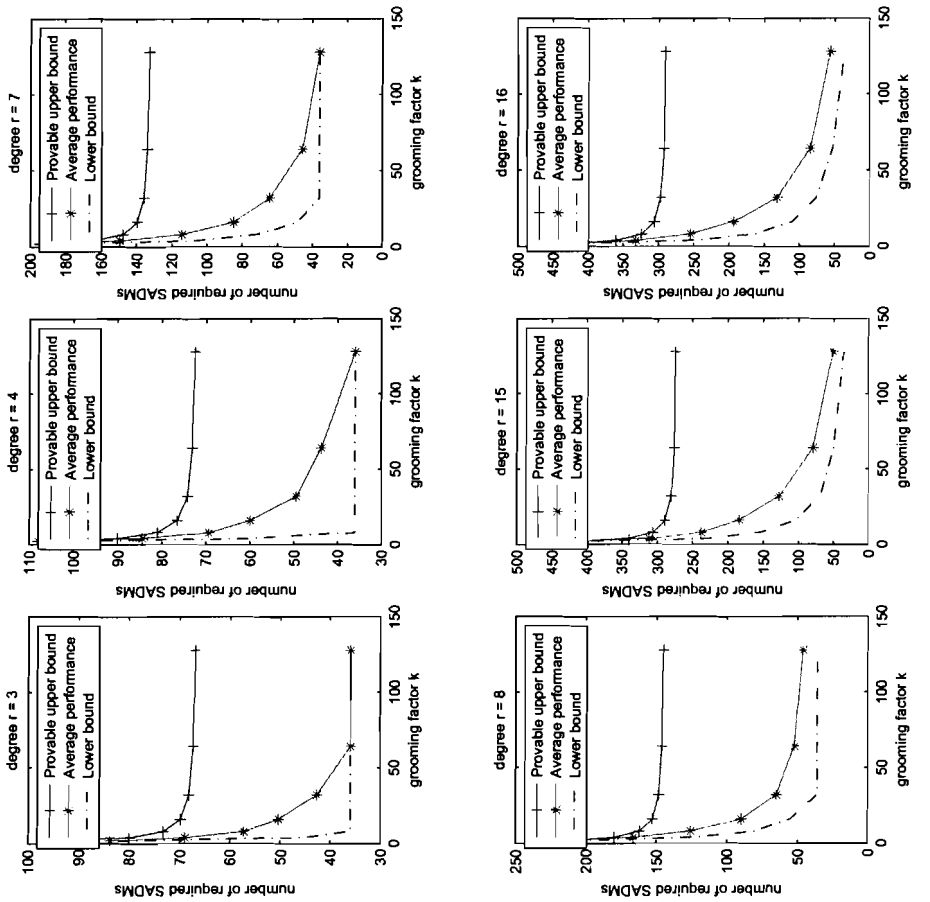

Figure 4.6: Average performance vs. worst case upper bound of Regular_Euler on graphs with $n = 36$ nodes.

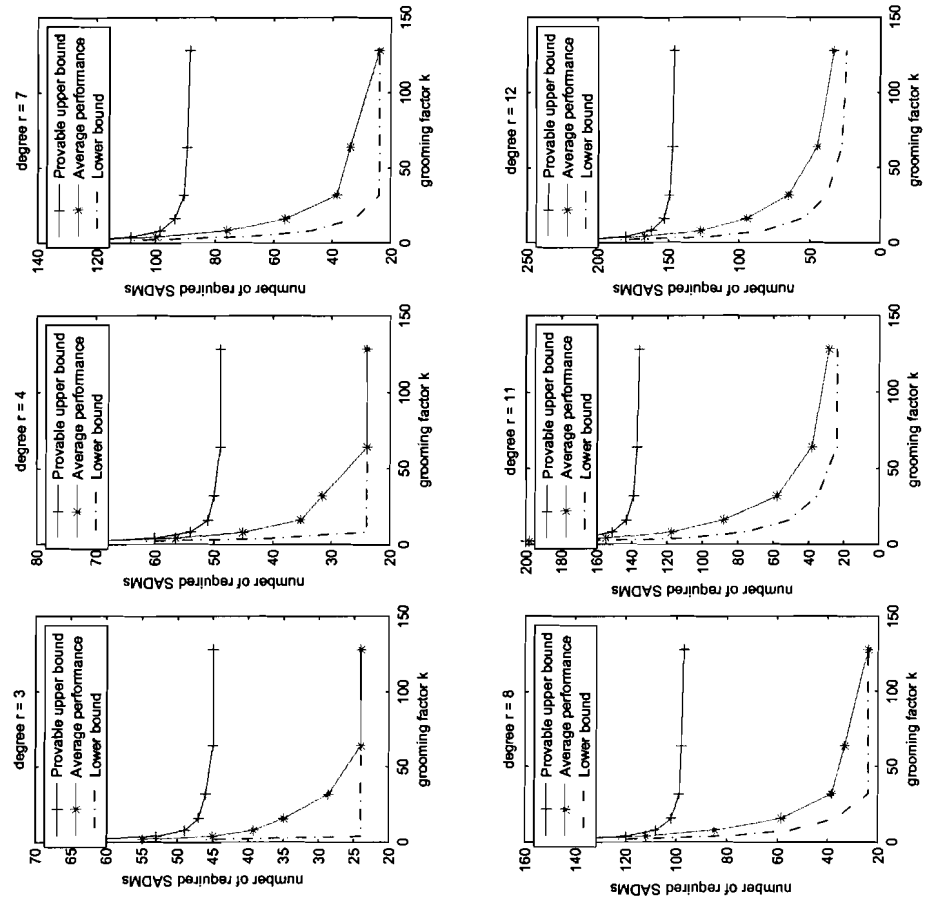

Figure 4.7: Average performance vs. worst case upper bound of Regular_Euler on graphs with $n = 24$ nodes.

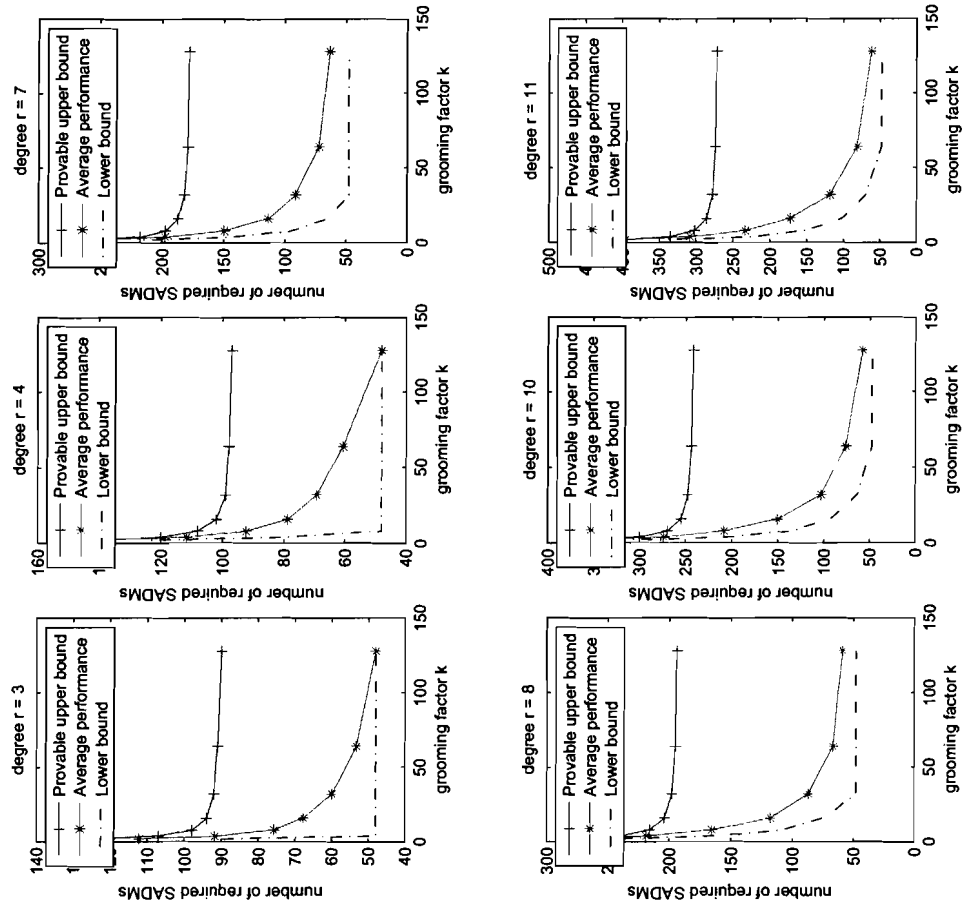

Figure 4.8: Average performance vs. worst case upper bound of Regular_Euler on graphs with $n = 48$ nodes.

Chapter 5

Min-Max and Max Throughput traffic grooming

In the previous two chapters, we have studied the traffic grooming problem to minimize the total number of used SADMs in order to satisfy the full connectivity for a given set of traffic demands. In this chapter, using novel graph partitioning approaches, we study traffic grooming from different point of views. We first consider a Min-Max optimization problem to minimize the maximum number of required SADMs at a network node over all network nodes. We focus on the Unidirectional Path-Switched Ring (UPSR) network with unitary duplex traffic demands. We prove the NP-hardness of this minimization problem, and propose a linear time $(\frac{k+1}{2} + 2)$-approximation algorithm. In addition, we study the problem to maximize the number of accommodated traffic demands in networks with limited number of SADMs at each network node. We prove that this maximization problem is NP-hard as well, and propose a $(k + 1)$-approximation algorithm. We also study the all-to-all traffic pattern, which is an important special case. For both the minimization and the maximization problems, we give algorithms achieving solutions only constant factors away from the optimal ones.

5.1 Problem formulation

We first consider the problem to minimize the maximum number of required SADMs over all network nodes. A similar Min-Max optimization goal has been discussed for the multi-hop traffic grooming problem [12]. Such a Min-Max traffic grooming problem is of high practical interest [12]. First of all, minimizing the maximum number of SADMs very likely produces a homogeneous network design in terms of the number of SADMs at each network node. Homogeneous nodes usually have lower cost than heterogeneous ones for network deployment, maintenance and upgrade due to the identical specifications. Furthermore, the Min-Max optimization goal is important for dealing with dynamic traffic demands. In particular, if the traffic pattern is changing dynamically within a given set, we can pre-compute a solution for each traffic pattern in the set, and deploy in every node with the largest number of SADMs required by any solution. Thus any traffic pattern from the given set can be satisfied without reconfiguring the deployment of SADMs.

We also study the dual problem of the above Min-Max optimization problem: maximize the throughput for a given set of traffic demands on a network with a limited number of SADMs at each network node. This problem is critical in the situation that there are no sufficient SADMs in the network to satisfy the full connectivity of the traffic demands. Similar sparse grooming networks are studied in [35, 62, 63, 69], where limited number of grooming resources are deployed unevenly among network nodes in the network. In this chapter, we consider homogeneous networks with limited number of grooming resources. Specifically, we consider the network in which every network node is equipped with a limited number L of SADMs, and we study the problem to maximize the number of satisfied traffic demands. Such a Maximum Throughput traffic grooming problem is natural in the WDM network operation: when the deployed SADMs are not sufficient to satisfy the full connectivity for a given set of traffic demands, a grooming scheme which satisfies as many traffic demands as possible is greatly desired.

Recall that for the unitary duplex traffic grooming problem in the UPSR network, a k-Edge-Partitioning problem has been formulated to minimize the total number of used SADMs, such that the full connectivity for a given set of traffic demands can be satisfied. We use a similar graph partition approach for the Min-Max traffic grooming problem. The Min-Max traffic grooming problem can be formulated as the following Min-Max k-Edge-Partitioning problem (MMkEP) on the traffic graph:

Min-Max k-Edge-Partitioning Problem (MMkEP)
Instance: A traffic graph $G(V, E)$ and integer $k \leq |E(G)|$.
Objective: Partition the edge set $E(G)$ into a collection of pairwise disjoint subsets $\mathcal{E} = \{E_1, E_2, \ldots\}$ such that $|E_i| \leq k$ for each $E_i \in \mathcal{E}$. The objective is to minimize the value of $\max_{v \in V(G)} A(v)$, where $A(v)$ is the number of E_i's each of which contains at least one edge incident to node v.

It is noticed that integer k corresponds to the grooming factor, each subset E_i corresponds to a wavelength, and $A(v)$ corresponds to the number of used SADMs in node v.

Similarly, the Maximum Throughput traffic grooming problem can be formulated as the following Maximum Connectivity k-Edge-Partitioning problem (MaxCkEP) on the traffic graph:

Maximum Connectivity k-Edge-Partitioning Problem (MaxCkEP)
Instance: A traffic graph $G(V, E)$, an integer $k \leq |E(G)|$ and an integer L.
Objective: Find a collection $\mathcal{E} = \{E_1, E_2, \ldots\}$ of pairwise disjoint subsets of $E(G)$ such that $|E_i| \leq k$ for each $E_i \in \mathcal{E}$ and $A(v) \leq L$ for each $v \in V(G)$, where $A(v)$ is the number of E_i's each of which contains at least one edge incident to node v. The objective is to maximize $\sum_{E_i \in \mathcal{E}} |E_i|$.

It is noticed that integer L corresponds to the number of available SADMs at each node, and the other parameters have the same meaning as we described in the MMkEP problem.

5.2 Min-Max k-Edge-Partitioning Problem

In this section, we prove that the MMkEP problem is NP-hard, and propose a linear time $(\frac{k+1}{2} + 2)$-approximation algorithm, where k is the grooming factor. In addition, we show that the algorithm achieves the worst case lower bound for the MMkEP problem. As an important special case, the all-to-all traffic pattern attracts a lot of research attention as well. The all-to-all traffic pattern has been well studied for minimizing the total number of used SADMs [5, 6, 7, 8, 32, 42, 64]. For the all-to-all traffic pattern, the traffic graph is a complete graph. We prove a lower bound $\frac{n-1}{\sqrt{2k}}$ for the MMkEP problem on complete graph K_n, and propose an algorithm achieving an upper bound $\frac{n}{\sqrt{\frac{2}{3}\sqrt{2k}-2}} + \frac{3}{2}$, which is a

constant factor away from the lower bound. The lower bound is interpreted in the Min-Max traffic grooming problem as that for the all-to-all traffic pattern on a network of n nodes, at least $\frac{n-1}{\sqrt{2k}}$ SADMs are needed at a network node with the maximum number of SADMs for any solution. The upper bound says that our algorithm gives a solution which requires at most $\frac{n}{\sqrt{\frac{2}{3}\sqrt{2k}-2}} + \frac{3}{2}$ SADMs at any node. We also show that the upper bound can be systematically improved further for relatively large values of k.

5.2.1 NP-hardness

In this section we analyze the computational complexity of the MMkEP problem. We first prove that the problem is NP-hard, and then show that the problem does not admit an FPTAS unless $P = NP$. We consider the decision version of the MMkEP problem, and prove that it is NP-complete. The decision version of the MMkEP problem is stated as follows:

Min-Max k-Edge-Partitioning (MMkEP) Problem
Instance: An undirected graph $G(V, E)$, and integers k, L.
Question: Is there a partition of $E(G)$ into a collection of pairwise disjoint subsets $\mathcal{E} = \{E_1, E_2, \ldots\}$ such that

1. $|E_i| \leq k$ for each $E_i \in \mathcal{E}$, and
2. $\max_{v \in V(G)} A(v) \leq L$, where $A(v)$ is the number of E_i's each of which contains at least one edge incident to node v.

We prove that the MMkEP problem is NP-complete by a reduction from the regular graph version of the Edge-Partition into Triangles (EPT) problem. The regular graph version of the EPT problem is proved to be NP-complete in Chapter 4 and stated as follows:

Edge-Partition into Triangles (EPT) Problem (regular graph version)
Instance: An undirected r-regular graph $G(V, E)$ with $|E(G)| = m$.
Question: Is there a partition of $E(G)$ into sets $\{E_1, E_2, ..., E_{m/3}\}$ such that each E_i induces a triangle?

Theorem 23 *The MMkEP problem is NP-complete.*

Proof: The MMkEP problem is clearly in NP. Given an instance of the regular graph version of the EPT problem with an undirected input r-regular graph G, we construct an instance of the MMkEP problem with G as the input graph, $k = 3$, and $L = \frac{r}{2}$. We prove that G can be partitioned into triangles if and only if there exists a solution for the MMkEP problem on the constructed instance.

If G can be partitioned into triangles, it is easy to verify that the triangle partition of G gives a solution for the MMkEP problem on the constructed instance.

We now prove that if the constructed instance has a solution $\mathcal{E} = \{E_1, E_2, \ldots\}$ for the MMkEP problem, then G can be partitioned into triangles. For solution $\mathcal{E}$ we have

1. $|E_i| \leq 3 = k$ for each $E_i \in \mathcal{E}$, and
2. $\max_{v \in V(G)} A(v) \leq \frac{r}{2} = L$.

Let G_i denote the sub-graph (of G) induced by each edge set $E_i \in \mathcal{E}$. We notice that $|V(G_i)| \geq |E(G_i)|$ since each graph G_i contains at most 3 edges, where the equation holds if and only if G_i is a triangle (i.e., the complete graph with three edges). Hence we have

$$\sum_{E_i \in \mathcal{E}} |V(G_i)| \geq \sum_{E_i \in \mathcal{E}} |E_i| = |E(G)|.$$

Therefore

$$\sum_{v \in V(G)} A(v) = \sum_{E_i \in \mathcal{E}} |V(G_i)| \geq |E(G)| = \frac{r|V(G)|}{2}.$$

We also know that $\max_{v \in V(G)} A(v) \leq \frac{r}{2}$, which implies that

$$\sum_{v \in V(G)} A(v) \leq \frac{r \cdot |V(G)|}{2}.$$

Therefore it must be the case that

$$\sum_{v \in V(G)} A(v) = \frac{r \cdot |V(G)|}{2},$$

which implies that every $E_i \in \mathcal{E}$ induces a triangle. That is, graph G can be partitioned into triangles. □

We further prove that the MMkEP problem does not admit an FPTAS unless $P = NP$.

Theorem 24 *The MMkEP problem does not admit an FPTAS unless $P = NP$.*

Proof: Assume that the MMkEP problem does admit an FPTAS. Now we construct an algorithm A that calls the FPTAS with precision $\epsilon = \frac{1}{r+1}$, where r is the maximum degree of graph G. Since the time complexity of an FPTAS is polynomial in the input size and also polynomial in $\frac{1}{\epsilon} = r + 1$ and $r + 1$ is at most $O(n)$, algorithm A is polynomial in the input size (i.e., an polynomial time algorithm for MMkEP). Given any input instance I, let $A(I)$ denote the value of the solution by algorithm A and $OPT(I)$ denote the optimum. Since any solution to the problem must be an integer, we assume $A(I) = OPT(I) + i$ for some positive integer i. We have $OPT(I) + i = A(I) \leq (1 + \epsilon)OPT(I) = (1 + \frac{1}{r+1})OPT(I)$, which implies that $OPT(I) \geq i(r + 1)$. It is observed that $OPT(I) \leq r$ for any instance, therefore $OPT(I) \geq i(r + 1)$ holds only when $i = 0$, which implies that $OPT(I) = A(I)$. So A is an optimal algorithm with polynomial running time, however this can not be true for the NP-hard problem MMkEP unless $P = NP$. □

5.2.2 A linear time $(\frac{k+1}{2} + 2)$-approximation algorithm

As the MMkEP problem is NP-hard, we will put our efforts on algorithms achieving approximate solutions instead of optimal solutions. In this section, we propose a linear time $(\frac{k+1}{2} + 2)$-approximation algorithm MinMax_Grooming for the MMkEP problem, where k is the grooming factor.

For a traffic graph G, we first construct a bipartite graph $G_b(U, V, E)$ based on G as follows:

1. For every edge $e_i \in E(G)$, there is a corresponding node u'_i in $U(G_b)$;
2. For every node $v_i \in V(G)$, there is a corresponding node v'_i in $V(G_b)$;
3. In graph G, if the edge corresponding to a node $u' \in U(G_b)$ is incident to the node corresponding to a node $v' \in V(G_b)$, then there exists an edge $(u', v') \in E(G_b)$.

Figure 5.1 shows an example of constructing bipartite graph G_b based on G. It is noticed that every node in $U(G_b)$ has degree two, and every node in $V(G_b)$ has the same degree as its corresponding node in graph G. Since the number of odd-degree nodes in any graph must be even, the number of odd-degree nodes in $V(G_b)$ is even.

We assume that every edge of graph G_b is unmarked initially, and we traverse and mark the edges of graph G_b as follows:

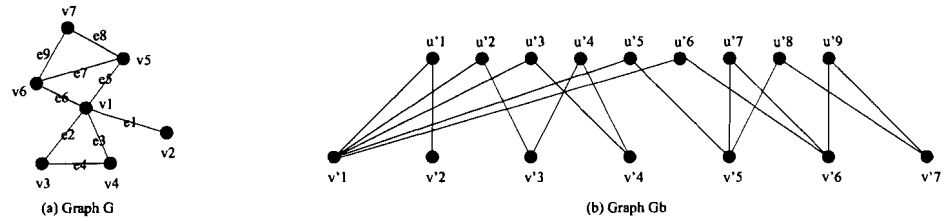

Figure 5.1: An example: graph G and the corresponding bipartite graph G_b.

1. If there exists a node in $V(G_b)$ that has odd number of unmarked adjacent edges, we start from such a node and traverse along arbitrary unmarked edges until arriving at a node with all adjacent edges marked, where after traversing each edge we immediately mark the edge as black if the edge is traversed from a node in $V(G_b)$ to a node in $U(G_b)$, and mark the edge as white otherwise. We repeat the above process until every node in $V(G_b)$ has even number of unmarked adjacent edges. It is observed we always start from an odd-degree node and terminate at an odd-degree node in $V(G_b)$ in each iteration;

2. If there still exist unmarked edges in G_b, we start from a node in $V(G_b)$ with unmarked adjacent edges, and traverse along unmarked edges until arriving at a node with all adjacent edges marked, where the traversed edges are marked the same way as above. We repeat until all the edges in G_b are marked. It is observed we terminates at the same node from which we start in each iteration.

Figure 5.2 shows a possible marking for the edges of the bipartite graph G_b constructed in Figure 5.1 according to the following traversing order:

1. Start from node v'_1, traverse along unmarked edges $\{v'_1, u'_1\}$,$\{u'_1, v'_2\}$ and mark each edge accordingly immediately after it is traversed, and terminate at node v'_2;

2. Start from node v'_5, traverse along unmarked edges $\{v'_5, u'_8\}$, $\{u'_8, v'_7\}$, $\{v'_7, u'_9\}$, $\{u'_9, v'_6\}$, $\{v'_6, u'_7\}$, $\{u'_7, v'_5\}$, $\{v'_5, u'_5\}$, $\{u'_5, v'_1\}$, $\{v'_1, u'_6\}$, $\{u'_6, v'_6\}$ and mark each edge accordingly immediately after it is traversed, and terminate at node v'_6;

3. Start from node v'_1, traverse along unmarked edges $\{v'_1, u'_3\}$, $\{u'_3, v'_4\}$, $\{v'_4, u'_4\}$, $\{u'_4, v'_3\}$, $\{v'_3, u'_2\}$, $\{u'_2, v'_1\}$ and mark each edge accordingly immediately after it is traversed,

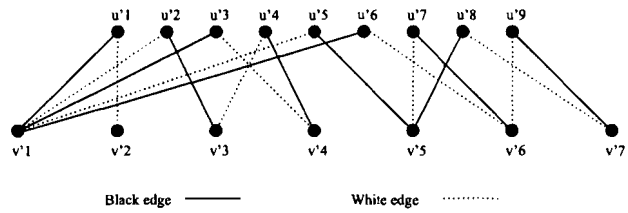

Figure 5.2: An example: Marking edges of G_b.

and terminate at node v'_1.

After traversing and marking all edges of graph G_b, the following properties hold:

1. Each node $v' \in V(G_b)$ is adjacent to at least $\lfloor \frac{\delta(v')}{2} \rfloor$ black edges and at least $\lfloor \frac{\delta(v')}{2} \rfloor$ white edges, where $\delta(v')$ is the degree of v'.

2. Each node $u' \in U(G_b)$ is adjacent to exactly one black edge and one white edge.

For every node $v' \in V(G_b)$, let $B_{v'} \subseteq U(G_b)$ denote the set of nodes which are adjacent to node v' via black edges, and let $E_{B_{v'}} \subseteq E(G)$ denote the set of edges which are corresponding to the nodes in $B_{v'}$. Since each node in $U(G_b)$ is adjacent to exactly one black edge, we have $\bigcup_{v' \in V(G_b)} B_{v'} = U(G_b)$, and $B_{v'_i} \cap B_{v'_j} = \emptyset$ for $v'_i \neq v'_j$. So we can conclude that $\bigcup_{v' \in V(G_b)} E_{B_{v'}} = E(G)$, and $E_{B_{v'_i}} \cap E_{B_{v'_j}} = \emptyset$ for $v'_i \neq v'_j$. Then for every node $v' \in V(G_b)$, we arbitrarily divide $E_{B_{v'}}$ into groups, each of which contains exactly k edges of graph G (except the last group might contain less than k edges). It is noticed that the edges in each group induce a star of graph G, where the center of the star is the node (of G) corresponding to node $v' \in G_b$. Therefore, we can construct a solution $\mathcal{E} = \{E_1, E_2, \ldots\}$ for the MMkEP problem on graph G, such that each $E_i \in \mathcal{E}$ is a group of edges as described above. Since each node $v' \in V(G_b)$ is adjacent to at least $\lfloor \delta(v')/2 \rfloor$ black edges, for solution $\mathcal{E}$ we have

$$\max_{v \in V(G)} A(v) \leq \lceil \frac{\lfloor \Delta(G)/2 \rfloor}{k} \rceil + \lceil \Delta(G)/2 \rceil,$$

where $\Delta(G)$ is the maximum degree of graph G. The pseudo code of Algorithm Min-Max_Grooming is given in Figure 5.3.

Theorem 25 *Algorithm MinMax_Grooming is a* $(\frac{k+1}{2} + 2)$*-approximation algorithm for the MMkEP problem.*

Algorithm MinMax_Grooming
Input: An undirected traffic graph G and integer k.
Output: A solution $\mathcal{E}$ for the MMkEP problem.
begin
Construct bipartite graph G_b based on graph G;
Traverse and mark edges in $E(G_b)$ as black and white edges;
Construct $\mathcal{E}$ according to the marking of $E(G_b)$.
end.

Figure 5.3: Pseudo code for Algorithm MinMax_Grooming.

Proof: For a traffic graph G and a grooming factor k, it is obvious that $\lceil \frac{\Delta(G)}{k} \rceil$ is a lower bound on any optimal solution of the MMkEP problem. Therefore the approximation ratio of Algorithm MinMax_Grooming is given by

$$\frac{\lceil \frac{\lfloor \frac{\Delta(G)}{2} \rfloor}{k} \rceil + \lceil \frac{\Delta(G)}{2} \rceil}{\lceil \frac{\Delta(G)}{k} \rceil} \leq \frac{\frac{\frac{\Delta(G)}{2}}{k} + \frac{\Delta(G)}{2} + 2}{\lceil \frac{\Delta(G)}{k} \rceil} \leq \frac{k+1}{2} + 2.$$

□

It is worth pointing out that there exist instances for which the optimal solution of the MMkEP problem is $\lceil \frac{\lfloor \Delta(G)/2 \rfloor}{k} \rceil + \lceil \Delta(G)/2 \rceil$, that is, Algorithm MinMax_Grooming achieves the worst case lower bound.

Theorem 26 *Algorithm MinMax_Grooming achieves the worst case lower bound of the MMkEP problem.*

Proof: Consider an instance of the MMkEP problem where $k = 2$ and the traffic graph G is a r-regular graph. Since G is a r-regular graph, there are $\frac{|V(G)| \cdot r}{2}$ edges in graph G. For any solution $\mathcal{E} = \{E_1, E_2, \ldots\}$, we have

$$\sum_{v \in V(G)} A(v) \geq \frac{|V(G)| \cdot r}{2} \cdot \frac{3}{2} = \frac{3}{4}|V(G)|r,$$

where the equation holds if and only if each $E_i \in \mathcal{E}$ is a path of length 2. So we have $\max_{v \in V(G)} A(v) \geq \frac{3}{4}r$, which implies that $\max_{v \in V(G)} A(v) \geq \lceil \frac{3}{4}r \rceil$ since $\max_{v \in V(G)} A(v)$ must be an integer. It is easy to verify that

$$\lceil \frac{3}{4}r \rceil = \lceil \frac{\lfloor r/2 \rfloor}{k} \rceil + \lceil r/2 \rceil \text{ when } k = 2,$$

hence Algorithm MinMax_Grooming achieves the worst case lower bound. □

The bipartite graph construction and the edge traversing/marking procedure can be done in $O(|E(G)|)$ time, and the solution based on the marking of $E(G_b)$ can be obtained in $O(|E(G)|)$ time as well. So Algorithm MinMax_Grooming runs in $O(|E(G)|)$ time, which is linear in the size of the input graph.

5.2.3 Special case: all-to-all traffic pattern

For the all-to-all traffic pattern, the traffic graph is a complete graph. We consider a complete traffic graph K_n containing n nodes.

A lower bound

We first prove a lower bound for the MMkEP problem on an arbitrary traffic graph.

Theorem 27 *For a traffic graph G, $\frac{\sqrt{2}|E(G)|}{|V(G)|\sqrt{k}}$ is a lower bound on optimal solutions for the MMkEP problem.*

Proof: It is noticed that the minimum value of $\sum_{v\in V(G)} A(v)$ is achieved if we can partition G into cliques, each of which contains exactly k edges. We also know that a clique of k edges contains $\frac{\sqrt{8k+1}+1}{2}$ nodes. Therefore we have

$$\sum_{v\in V(G)} A(v) \geq \frac{|E(G)|}{k} \cdot \frac{\sqrt{8k+1}+1}{2},$$

and thus

$$\max_{v\in V(G)} A(v) \geq \frac{|E(G)|}{|V(G)|\cdot k} \cdot \frac{\sqrt{8k+1}+1}{2} > \frac{\sqrt{2}|E(G)|}{|V(G)|\sqrt{k}}.$$

□

Corollary 28 *For complete graph K_n, $\frac{n-1}{\sqrt{2k}}$ is a lower bound on optimal solutions for the MMkEP problem.*

For the case that grooming factor $k = 2$, since K_n is a regular graph as well, the proof of Theorem 26 shows that Algorithm MinMax_Grooming achieves the optimal solution. In the following we assume that $k \geq 3$.

Upper bounds

For complete graph K_n, we propose an algorithm achieving a solution which is a constant factor away from the above lower bound. The basic idea of the algorithm is to balance the number of required SADMs among all nodes. In order to do so, we construct a complete graph K_N based on k_n such that each node in K_N corresponds to a set of g nodes in K_n, and each edge in K_N corresponds to the set of edges between the two corresponding sets of nodes in K_n. Then using the results from design theory, we partition K_N into triangles. For each of such triangles, the three edges corresponds to a set of edges in K_n, which constitutes a sub-graph in the solution of the MMkEP problem on K_n if we choose the value of g to be $\lfloor\sqrt{\frac{k}{3}}\rfloor$.

Theorem 29 *A solution with* $\max_{v\in V(K_n)} A(v) \leq \frac{n}{\sqrt{\frac{2}{3}\sqrt{2k}-2}} + \frac{3}{2}$ *can be obtained for the MMkEP problem on complete graph* K_n.

Proof: Let $g = \lfloor\sqrt{\frac{k}{3}}\rfloor$ and we divide the n nodes of K_n into $\lceil n/g\rceil$ groups, each of which contains g nodes of K_n. To simplify the description, we assume the last group also contains g nodes even it might contain less than g nodes.

Let N be the smallest integer satisfying $N \geq \lceil n/g\rceil$ and $N \equiv 1, 3 \pmod 6$. We construct a complete graph K_N in the following, where we call each node of K_N a *super node*, and each edge of K_N a *super edge*:

1. for each of the $\lceil n/g\rceil$ node groups in K_n, add a corresponding super node in K_N,
2. if $N > \lceil n/g\rceil$, add $N - \lceil n/g\rceil$ virtual super nodes in K_N, and
3. add a super edge between each pair of super nodes in K_N,

where each super node in K_N corresponds to a clique of size g in K_n, and each super edge corresponds to $g^2 \leq \frac{k}{3}$ edges that are between nodes of two corresponding cliques in K_n.

Using results from the design theory, Bermond and Ceroi [5] showed that K_N can be partitioned into $\frac{N(N-1)}{6}$ $K_3's$ (i.e., triangles) if $N \equiv 1, 3 \pmod 6$. For each triangle, the three super edges correspond to $3g^2 = 3\lfloor\sqrt{\frac{k}{3}}\rfloor^2 \leq k$ edges in K_n, therefore we can put each set of such $3g^2$ edges into one subset E_i in a solution $\mathcal{E}$ of the MMkEP problem on K_n. For each super node in K_N, there are $\frac{g(g-1)}{2} < \frac{k}{6}$ edges in the corresponding clique of K_n,

therefore we can put $3g(g-1)$ edges from each 6 corresponding cliques into one subset E_i in solution $\mathcal{E}$. It is clear for the above solution $\mathcal{E}$ we have

$$\max_{v \in V(K_n)} A(v) = \frac{N-1}{2} + 1 \leq \frac{n}{\sqrt{\frac{2}{3}}\sqrt{2k} - 2} + \frac{3}{2}.$$

□

The above upper bound can be improved further for relatively large values of k. Bermond *et al.* showed that K_N can be partitioned into K_4's if $N \equiv 1, 4 \pmod{12}$, can be partitioned into K_5's if $N \equiv 1, 5 \pmod{20}$, and can be partitioned into K_6's if $N \equiv 1 \pmod{30}$ [8]. So for the case of $k \geq 6$, we can divide the nodes of K_n into groups, each of which has $\lfloor\sqrt{\frac{k}{6}}\rfloor$ nodes. Then each group is represented by a super node in K_N, and minimum number of virtual super nodes are added such that the total number of super nodes N satisfies that $N \equiv 1, 4 \pmod{12}$. So complete graph K_N on N super nodes can be partitioned into K_4's. Similar as described above, the K_4 partition of K_N can be transferred into a solution $\mathcal{E}$ for the MMkEP problem on K_n with

$$\max_{v \in V(K_n)} A(v) = \frac{N-1}{3} + 1 \leq \frac{n}{\sqrt{\frac{3}{4}}\sqrt{2k} - 3} + \frac{10}{3}.$$

Bermond and Coudert [8] also have proved that K_N can be partitioned into K_5's if $N \equiv 1, 5 \pmod{20}$, and partitioned into K_6's if $N \equiv 1 \pmod{30}$. The similar idea can be used to further improve the above upper bounds, and accordingly the upper bounds that we can obtain are

$$\frac{n}{\sqrt{\frac{4}{5}}\sqrt{2k} - 4} + \frac{9}{2} \text{ for } k \geq 10, \text{ and } \frac{n}{\sqrt{\frac{5}{6}}\sqrt{2k} - 5} + \frac{33}{5} \text{ for } k \geq 15$$

respectively.

5.3 Maximum Connectivity k-Edge-Partitioning Problem

In this section, we prove that the MaxCkEP problem is NP-hard as well, and propose a $(k+1)$-approximation algorithm. We also prove an upper bound $\frac{nL\sqrt{k}}{\sqrt{2}}$ for a traffic graph with n nodes. This upper bound indicates that any solution of the MaxCkEP problem can accommodate at most $\frac{nL\sqrt{k}}{\sqrt{2}}$ duplex demands. For the special case of complete traffic graph K_n, we propose an algorithm achieving a lower bound $\frac{nL\lfloor\sqrt{k}\rfloor}{2}$. This lower bound states that the number of accommodated duplex demands by our algorithm is at least $\frac{nL\lfloor\sqrt{k}\rfloor}{2}$, which is only a constant factor (about $\sqrt{2}$) away from the upper bound and thus the optimum.

5.3.1 NP-hardness

In this section, we prove the NP-hardness of the MaxCkEP problem. We consider the decision version of the problem, and prove that it is NP-complete. The decision version of the MaxCkEP problem is stated as follows:

Maximum Connectivity k-Edge-Partitioning (MaxCkEP) Problem
Instance: An undirected graph $G(V, E)$, and integers k, L, T.
Question: Is there a collection $\mathcal{E} = \{E_1, E_2, \ldots\}$ of pairwise disjoint subsets of $E(G)$ such that

1. $|E_i| \leq k$ for each $E_i \in \mathcal{E}$,
2. $A(v) \leq L$ for each $v \in V(G)$, where $A(v)$ is the number of E_i's each of which contains at least one edge incident to node v, and
3. $\sum_{E_i \in \mathcal{E}} |E_i| \geq T$.

Theorem 30 *The MaxCkEP problem is NP-complete.*

Proof: We prove that the MaxCkEP problem is NP-complete by a reduction from the MMkEP problem. The MaxCkEP problem is clearly in NP. Given an instance of the MMkEP problem with an undirected input graph G, integers k and L, we construct an instance of the MaxCkEP problem with the same input graph G, the same integers k and L, and $T = |E(G)|$. It is easy to verify that the MMkEP problem has a solution if and only if the MaxCkEP problem has a solution. □

5.3.2 A $(k+1)$-approximation algorithm

In this section, we propose a $(k+1)$-approximation algorithm MaxConnectivity_Grooming for the MaxCkEP problem, where k is the grooming factor.

For a traffic graph G, we first introduce a rather simple approach to compute solutions for the MaxCkEP problem. Let M_L be a maximum b-matching of G with $b(v) = L$ for every $v \in V(G)$. Since each node $v \in V(G)$ appears in at most L edges of M_L, a feasible solution for the MaxCkEP problem can be obtained by arbitrarily partitioning M_L into disjoint subsets, each of which contains at most k edges. We use $\mathcal{E}_{M_L}$ to denote such a solution.

Now we propose the following algorithm MaxConnectivity_Grooming to compute a solution $\mathcal{E}$ for the MaxCkEP problem on the traffic graph G. Initially $\mathcal{E}$ is an empty set. For each $v \in V(G)$, we define *capacity(v)*$= L - A(v)$ (i.e., the number of available SADMs in node v). At the beginning of the algorithm, $A(v) = 0$ since $\mathcal{E}$ is empty, and capacity$(v) = L$ for every node $v \in V(G)$. Algorithm MaxConnectivity_Grooming runs in iterations. For each iteration, a star of k edges is constructed and the edges in the star constitute a subset E_i in the solution $\mathcal{E}$. Then the star is removed from graph G, and the degree and capacity of each node is updated accordingly. Intuitively, a node with higher degree and lower capacity should be given higher priority to be chosen as the center of the star. We also notice that once a star of k edges is removed from G, the capacity of the center is decreased by 1 and the degree of the center is decreased by k. Therefore we define

$$rank(u) = \delta(u) - k \cdot capacity(u).$$

For each iteration we choose a node u with the largest value $rank(u)$ as the center of the star (in the case of ties, we arbitrarily break the tie). Then the star is constructed by arbitrarily picking $\min(k, \delta(u))$ edges incident to node u as the edge set. After a star is constructed and removed from graph G, if there exists a node u with $capacity(u) = 0$, graph G is updated by deleting all edges incident to node u. The same procedure is repeated until all edges in graph G are removed. Then the solution $\mathcal{E}$ is compared to solution $\mathcal{E}_{M_L}$, which is obtained by the b-matching based approach mentioned above. If $\mathcal{E}_{M_L}$ is a better solution than $\mathcal{E}$, then $\mathcal{E}$ is simply replaced by $\mathcal{E}_{M_L}$. The pseudo code of Algorithm MaxConnectivity_Grooming is given in Figure 5.4.

Theorem 31 *Algorithm MaxConnectivity_Grooming is a $(k+1)$-approximation algorithm for the MaxCkEP problem.*

Proof: Let M_{kL} be a maximum b-matching of graph G with $b(v) = kL$ for every node $v \in V(G)$. Consider an optimal solution $\mathcal{E}_{opt}$ for the MaxCkEP problem on graph G. We first prove that $|M_{kL}|$ is an upper bound on solution $\mathcal{E}_{opt}$ (i.e., $\sum_{E_i \in \mathcal{E}_{opt}} |E_i| \leq |M_{kL}|$). Define $G_{\mathcal{E}_{opt}}$ as the graph induced by all the edges in $\bigcup_{E_i \in \mathcal{E}_{opt}} E_i$. We observe that the maximum degree $\Delta(G_{\mathcal{E}_{opt}}) \leq kL$, since the initial capacity of each node is L and one unit of capacity can be used for at most k edges incident to the node. So the edge set $E(G_{\mathcal{E}_{opt}})$ is a b-matching of graph G with $b(v) = kL$, and we can conclude that

$$\sum_{E_i \in \mathcal{E}_{opt}} |E_i| = |E(G_{\mathcal{E}_{opt}})| \leq |M_{kL}|.$$

Algorithm MaxConnectivity_Grooming
Input: An undirected traffic graph G, integer k and integer L.
Output: A solution $\mathcal{E}$ for the MaxCkEP problem.
begin
$\mathcal{E} := \emptyset$;
capacity$(v) := L$ for every node $v \in V(G)$;
While $|E(G)| \neq 0$ **do**
{Pick a node $u \in G$ with the largest value of *rank*(u);
Construct star S with u as center, and $|E(S)| = \min(k, \delta(u))$;
Update $\mathcal{E} := \mathcal{E} \cup \{E(S)\}$;
Update graph G by removing $E(S)$ from G,
and decreasing *capacity*(v) by 1 for every $v \in V(S)$;
For any $v \in V(S)$ with *capacity*$(v) = 0$ **do**
update graph G by removing edges incident to v;
};
Compute a maximum b-matching M_L ($b(v) = L$ for every $v \in V(G)$);
Construct solution $\mathcal{E}_{M_L}$ based on M_L;
If $\sum_{E_i \in \mathcal{E}} |E_i| < \sum_{E_i \in \mathcal{E}_{M_L}} |E_i|$ **then** $\mathcal{E} := \mathcal{E}_{M_L}$;
end.

Figure 5.4: Pseudo code for Algorithm MaxConnectivity_Grooming.

Let M_L be a maximum b-matching of graph G with $b(v) = L$ for every node $v \in V(G)$. Consider a solution $\mathcal{E} = \{E_1, E_2, \ldots\}$ obtained by Algorithm MaxConnectivity_Grooming for the MaxCkEP problem on graph G. It is clear that $|M_L|$ is a lower bound on solution $\mathcal{E}$, that is, $\sum_{E_i \in \mathcal{E}} |E_i| \geq |M_L|$.

Let $G_{M_{kL}}$ be the graph induced by edges in a maximum b-matching of G with $b(v) = kL$ for each $v \in V(G)$. By the definition, we know that the maximum degree $\Delta(G_{M_{kL}}) \leq kL$. For such a graph, we can use $(k+1)$ colors to obtain an f-coloring with $f(v) = L$ for each $v \in V(G_{M_{kL}})$. Then each set of edges with the same color in the f-coloring is a b-matching of G with $b(v) = L$. Hence, we can conclude that $(k+1)|M_L| \geq |M_{kL}|$. Therefore the approximation ratio of Algorithm MaxConnectivity_Grooming is given by

$$\frac{\sum_{E_i \in \mathcal{E}_{opt}} |E_i|}{\sum_{E_i \in \mathcal{E}} |E_i|} \leq \frac{|M_{kL}|}{|M_L|} \leq k+1.$$

□

For the star extraction part of Algorithm MaxConnectivity_Grooming, a set of edges are removed from the graph in each iteration. The process is repeated until all the edges are removed. Therefore, the running time is $O(|E(G)|)$ time, which is linear in the size of

the input graph. While for the part to compute a maximum b-matching with $b(v) = L$, the running time is $\sqrt{L|V(G)|}|E(G)|$ [23]. Therefore, Algorithm MaxConnectivity_Grooming runs in $\sqrt{L|V(G)|}|E(G)|$ time.

5.3.3 Special case: all-to-all traffic pattern

In this section, we study the MaxCkEP problem on complete traffic graph K_n containing n nodes, each of which has a capacity of L initially.

An upper bound

We first prove an upper bound for the MaxCkEP problem on an arbitrary traffic graph, in which each node has a capacity L initially.

Theorem 32 *For a traffic graph G, $\frac{|V(G)|L\sqrt{k}}{\sqrt{2}}$ is an upper bound on optimal solutions for the MaxCkEP problem.*

Proof: For graph G, the total capacity over all nodes is $|V(G)|L$. It is observed that a clique of k edges consumes the minimum units of capacity, which is $\frac{\sqrt{8k+1}+1}{2}$, among all sub-graphs of k edges. Therefore an upper bound on optimal solutions is

$$\frac{|V(G)|L}{\frac{\sqrt{8k+1}+1}{2}} \cdot k < \frac{|V(G)|L}{\frac{\sqrt{8k}}{2}} \cdot k = \frac{|V(G)|L\sqrt{k}}{\sqrt{2}}.$$

□

Corollary 33 *For complete graph K_n, $\frac{nL\sqrt{k}}{\sqrt{2}}$ is an upper bound on optimal solutions for the MaxCkEP problem.*

A lower bound

For complete graph K_n, we propose an algorithm achieving a solution which is a constant factor away from the above upper bound. We first prove the following lemma about a complete graph K_{2m} with even number $2m$ of nodes.

Lemma 34 *The edge set of complete graph K_{2m} can be partitioned into $(2m-1)$ pairwise disjoint subsets, such that each subset is a perfect matching of K_{2m}.*

Proof: It is known that an edge coloring of K_{2m} can be obtained using $2m-1$ colors [22]. We notice that each set of edges with the same color is a matching of K_{2m}, so the number of edges with the same color is at most m for every color. The total number of edges in K_{2m} is $m(2m-1)$, therefore it must be the case that the number of edges with the same color is exactly m for each color, which implies that each set of edges with the same color is a perfect matching of K_{2m}. □

The basic idea of the algorithm is still to balance the use of SADMs among all nodes. In order to do so, we construct a complete graph K_N based on k_n such that each node in K_N corresponds to a set of g nodes in K_n, and each edge in K_N corresponds to the set of edges between the two corresponding sets of nodes in K_n. Then we partition K_N into perfect matchings according to Lemma 34. Each edge in the matchings corresponds to a set of edges in K_n, which constitutes a sub-graph in the solution of the MaxCkEP problem on K_n if we choose the value of g to be $\lfloor\sqrt{k}\rfloor$.

Theorem 35 *A solution $\mathcal{E}$ with $\sum_{E_i\in\mathcal{E}}|E_i| \geq \frac{nL\lfloor\sqrt{k}\rfloor}{2}$ can be obtained for the MaxCkEP problem on complete graph K_n.*

Proof: Let $g = \lfloor\sqrt{k}\rfloor$ and we divide the n nodes of K_n into $\lceil n/g\rceil$ groups, each of which contains g nodes of K_n. To simplify the description, we assume that the last group also contains g nodes even it might contain less than g nodes.

Let $N = \lceil n/g\rceil$, and we construct a complete graph K_N, where we call each node of K_N a *super node*, and each edge of K_N a *super edge*:

1. for each of the $\lceil n/g\rceil$ node groups in K_n, add a corresponding super node in K_N, and
2. add a super edge between each pair of super nodes in K_N,

where each super node in K_N corresponds to a clique of size g in K_n, and each super edge corresponds to $g^2 \leq k$ edges that are between nodes of two corresponding cliques in K_n. We assume that N is even, since otherwise we can add one virtual super node and some corresponding virtual super edges.

According to Lemma 34, the edge set of K_N can be partitioned into $(N-1)$ perfect matchings. Since every super edge in each perfect matching corresponds to $g^2 \leq k$ edges of K_n, we can arbitrarily pick L perfect matchings, and construct solution $\mathcal{E} = \{E_1, E_2, \ldots\}$ such that each $E_i \in \mathcal{E}$ consists of edges (of K_n) corresponding to a super edge in the L

perfect matchings of K_N. Therefore

$$\sum_{E_i \in \mathcal{E}} |E_i| = L \cdot \frac{N}{2} \cdot g^2 = L \cdot \frac{\lceil n/g \rceil}{2} \cdot g^2 \geq \frac{nLg}{2} = \frac{nL\lfloor \sqrt{k} \rfloor}{2}.$$

□

The lower bound obtained above is based on two assumptions: the number of nodes in the last group is g, and N is an even integer. It can be easily verified that the difference will only be a constant value even if we do not make the assumptions.

5.4 Empirical results

In this section, we will conduct extensive simulations to evaluate the practical performances of our algorithms by comparing to the corresponding worst case performances and lower/upper bounds. We implemented Algorithm MinMax_Grooming and Algorithm MaxConnectivity_Grooming to validate their performances on randomly generated traffic graphs. In generating graphs, we specify the number n of nodes and density ratio d that is used to calculate the number $m = n^{1+d}$ of edges. A graph G of m edges and n nodes is generated by randomly connecting m pairs of nodes among all $n(n-1)/2$ possible pairs. Figure 5.5 shows the empirical results of Algorithm MinMax_Grooming on graphs of 36 nodes with density ratio $d = 0.6$, where for each value of grooming factor k, we collect the results from 100 randomly generated traffic graphs and compute the average value. The lower bound we used for measuring Algorithm MinMax_Grooming is $\lceil \frac{\Delta(G)}{k} \rceil$, which is mentioned in Theorem 25. Figure 5.8 shows the empirical results of Algorithm MaxConnectivity_Grooming on graphs of 36 nodes with density ratio $d = 0.6$ and grooming factor $k = 4$, where for each value of capacity L, we collect the results from 100 randomly generated traffic graphs and compute the average value. The upper bound we used for measuring Algorithm MaxConnectivity_Grooming is $|M_{kL}|$, which is mentioned in Theorem 31. We observe that the experimental performances of both algorithms are considerably better than the worst case performances proved in Section 5.2.2 and Section 5.3.2. In addition, the results show that the solution obtained by Algorithm MinMax_Grooming is close to the corresponding lower bound, and the solution obtained by Algorithm MaxConnectivity_Grooming is close to the corresponding upper bound. Simulations are conducted on graphs with different parameter settings as well, and the results present similar characteristics (see Figure 5.6, Figure 5.7, Figure 5.9, and Figure 5.10 for other representative results).

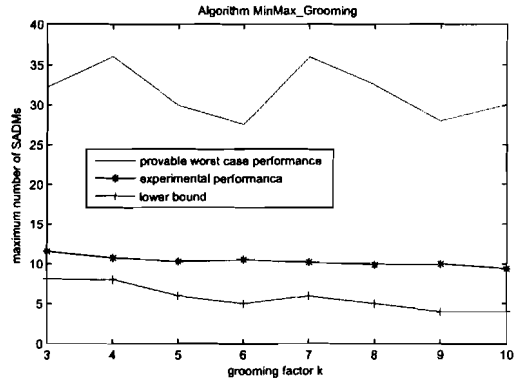

Figure 5.5: Performance of Algorithm MinMax_Grooming for $n = 36$ and $d = 0.6$.

5.5 Summary

In this chapter, using novel graph partitioning approaches, we studied traffic grooming for both the problem to minimize the maximum number of SADMs over all network nodes, and the problem to maximize the number of accommodated traffic demands subject to limited number of SADMs. We proved both problems are NP-hard for the UPSR network with unitary duplex traffic demands, and proposed an approximation algorithm for each problem. We also studied the all-to-all traffic pattern, and proposed algorithms achieving solutions only constant factors away from the optimal ones for both problems. For the future work, it is interesting to further narrow the gap between the lower bound and the upper bound for both the MMkEP problem and the MaxCkEP problem. Intuitively, a good approach for both problems should balance the usage of SADMs among all nodes. As we did in Algorithm MinMax_Grooming and Algorithm MaxConnectivity_Grooming, stars are extracted from the traffic graph in iterations to achieve such a balance. However, when the degrees of all nodes tend to be uniform (the extreme case is that all nodes have the same degree, i.e., the traffic graph is a regular graph), extracting a star each time from the traffic graph might not be good enough. A possible approach could be to extract a regular sub-graph for each iteration from the traffic graph. Therefore, the ideal solution for both problems should also take the structure of the traffic graph into account, and use a combination of star and other topologies for extracting sub-graphs.

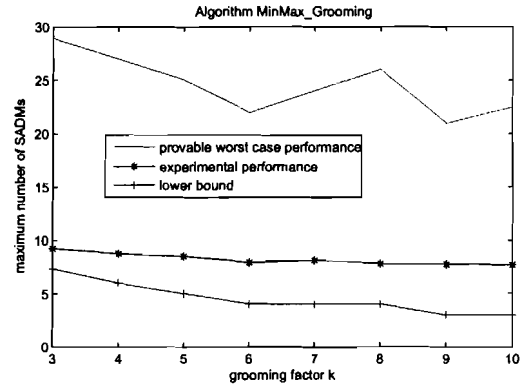

Figure 5.6: Performance of Algorithm MinMax_Grooming for $n = 48$ and $d = 0.5$.

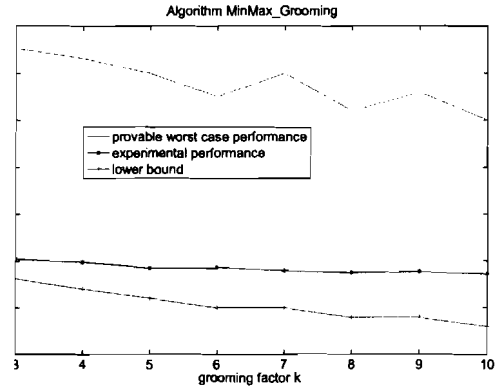

Figure 5.7: Performance of Algorithm MinMax_Grooming for $n = 60$ and $d = 0.5$.

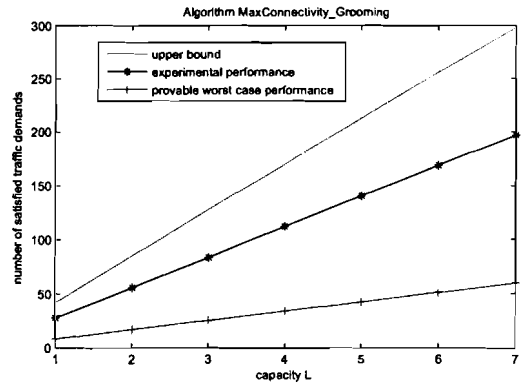

Figure 5.8: Performance of Algorithm MaxConnectivity_Grooming for $n = 36$, $k = 4$, and $d = 0.6$.

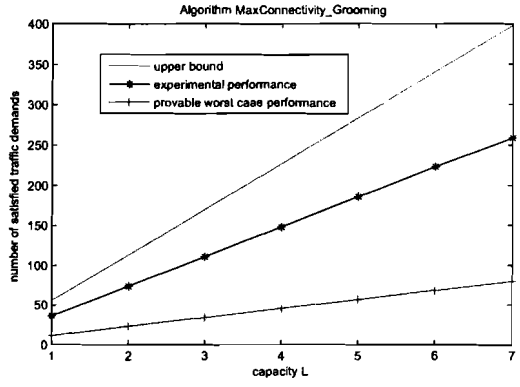

Figure 5.9: Performance of Algorithm MaxConnectivity_Grooming for $n = 48$, $k = 4$, and $d = 0.5$.

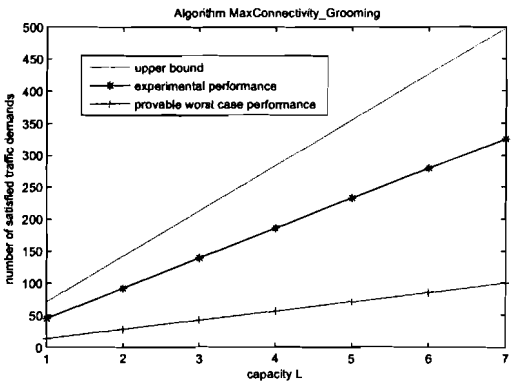

Figure 5.10: Performance of Algorithm MaxConnectivity_Grooming for $n = 60$, $k = 4$, and $d = 0.5$.

Chapter 6

Traffic grooming in BLSR networks

As another important architecture for SONET/WDM ring networks, BLSR is widely used in optical network design as well. Compared to UPSR networks, the routing of a traffic demand in BLSR networks is not unique. Traffic demands can be routed in either clockwise or counter-clockwise direction. Generally speaking, this flexibility in routing makes the traffic grooming problem in BLSR networks more complicated than UPSR networks. As mentioned in Chapter 2, the traffic grooming problem in BLSR network is usually solved by a two-step approach: the first step is to generate primitive cycles, and the second step is to groom primitive cycles into wavelength channels. The two steps are not completely independent, therefore it can not be guaranteed that the number of used SADMs is optimal even if the optimal solution can be achieved for each step individually. For the traffic grooming problem in BLSR networks, we adopt the two-step approach, and use previous algorithms to obtain the solution for the first step. From now on, we assume that the solution for the first step is available unless otherwise stated, and focus on solving the second step.

6.1 Definitions and problem formulations

We formulate the traffic grooming problems into hypergraph partitioning problems. A *hypergraph* $H(V, E_H)$ of n nodes and m hyperedges is a graph with node set V and hyperedge set $E_H = \{e_1, e_2, \ldots, e_m\}$, where each hyperedge e_i is a subset of V with n_i ($n_i \geq 2$) nodes.

For the two-step approach to solve the traffic grooming in BLSR networks, once the solution for the first step is given, we can construct a traffic hypergraph as follows: the node set represents the set of network nodes in the BLSR network; there is a hyperedge in the

hypergraph corresponding each primitive cycle, where the set of nodes in a hyperedge is the set of network nodes as the sources or destinations of the traffic demands groomed in the corresponding primitive cycle. We will formulate the traffic grooming problems using this hypergraph model.

Similar as the k-Edge-Partitioning Problem, the traffic grooming problem to minimize the total number of SADMs in BLSR networks can be formulated as the following partitioning problem on the traffic hypergraph:

k-Edge-Partitioning of Hypergraph Problem (kEPH)
Instance: A traffic hypergraph $H(V, E)$ and integer $k \leq |E(H)|$.
Objective: Partition the hyperedge set $E(H)$ into a collection of subsets $\mathcal{E} = \{E_1, E_2, \ldots\}$ (where $\bigcup_i E_i = E(H)$ and $E_p \cap E_q = \emptyset$ for $p \neq q$), such that $|E_i| \leq k$ for each $E_i \in \mathcal{E}$ and $\sum_{E_i \in \mathcal{E}} |V_i|$ is minimized, where V_i is the set of nodes in the sub-hypergraph induced by hyperedge set E_i.

Similar as the Min-Max k-Edge-Partitioning Problem, the Min-Max traffic grooming problem in BLSR networks can be formulated as the following Min-Max k-Edge-Partitioning of Hypergraph problem (MMkEPH):
Min-Max k-Edge-Partitioning of Hypergraph Problem (MMkEPH)
Instance: A traffic hypergraph $H(V, E)$ and integer $k \leq |E(H)|$.
Objective: Partition the hyperedge set $E(H)$ into a collection of pairwise disjoint subsets $\mathcal{E} = \{E_1, E_2, \ldots\}$ such that $|E_i| \leq k$ for each $E_i \in \mathcal{E}$. The objective is to minimize the value of $\max_{v \in V(H)} A(v)$, where $A(v)$ is the number of E_i's each of which contains at least one hyperedge incident to node v.

Similar as the Maximum Connectivity k-Edge-Partitioning Problem, the Maximum Throughput traffic grooming problem in BLSR networks can be formulated as the following Maximum Connectivity k-Edge-Partitioning of Hypergraph problem (MaxCkEPH):

Maximum Connectivity k-Edge-Partitioning of Hypergraph Problem (MaxCkEPH)
Instance: A traffic hypergraph $H(V, E)$, an integer $k \leq |E(H)|$ and an integer L.
Objective: Find a collection $\mathcal{E} = \{E_1, E_2, \ldots\}$ of pairwise disjoint subsets of $E(H)$ such that $|E_i| \leq k$ for each $E_i \in \mathcal{E}$ and $A(v) \leq L$ for each $v \in V(H)$, where $A(v)$ is the number

of E_i's each of which contains at least one hyperedge incident to node v. The objective is to maximize $\sum_{E_i \in \mathcal{E}} |E_i|$.

Since all the three corresponding graph partitioning problems are NP-hard, it can be easily proved that the following theorem holds.

Theorem 36 *The kEPH problem, the MMkEPH problem, and the MaxCkEPH problem are NP-hard.*

Proof: Trivial. □

6.2 Algorithms

In this section, we will focus on designing algorithms for the MMkEPH problem and the MaxCkEPH problem. As to the kEPH problem, it has been well studied in previous work which uses the hypergraph partitioning formulation implicitly.

6.2.1 Algorithms for the MMkEPH problem

In this section, we propose a simple greedy algorithm MinMax_Hyper for the MMkEPH problem. The basic idea of this algorithm is to balance the SADM usage among all the nodes in the network. The algorithm runs in iterations. For each iteration, a node v with the maximum degree is chosen. Then arbitrary k hyperedges incident to v are chosen to form a sub-hypergraph and deleted from the hypergraph, and accordingly the degrees of the affected nodes in the hypergraph are updated. The same process is repeated until every hyperedge is deleted from the hypergraph and included into some sub-hypergraph. During each iteration, we also maintain a variable $L(v)$ for each node v to denote the number of sub-hypergraphs containing at least one hyperedge incident to v (i.e., $L(v)$ denotes the number of consumed SADMs at node v) so far. Thus if there is a tie when comparing the degrees of the nodes, we choose the node with the largest $L(v)$ to break the tie (we will arbitrarily break the tie if it still exists). Eventually, all the sub-hypergraphs will constitute a hypergraph partition, which gives a solution for the MMkEPH problem. The pseudo code of Algorithm MinMax_Hyper is given in Figure 6.1. We will show the empirical results in Section 6.3.

Algorithm MinMax_Hyper
Input: A traffic hypergraph H and integer k.
Output: A solution $\mathcal{E}$ for the MMkEPH problem.
begin
$\mathcal{E} := \emptyset$;
While $|E(H)| \neq 0$ **do**
{Pick a node $v \in V(H)$ with the largest degree $\delta(v)$;
(break the tie by choosing a node with the largest $L(v)$)
Construct E_i consisting of $\min(k, \delta(v))$ hyperedges incident to v;
Update $\mathcal{E} := \mathcal{E} \cup \{E_i\}$;
Update hypergraph H by removing hyperedges in E_i;
Update degree $\delta(u)$ of each node u contained in the hyperedges of E_i;
Update $L(u)$ of each node u contained in the hyperedges of E_i;
};
end.

Figure 6.1: Pseudo code for Algorithm MinMax_Hyper.

During each iteration of the algorithm, a set of hyperedges are removed from the hypergraph. The process is repeated until all the hyperedges are removed. Therefore, Algorithm MinMax_Hyper runs in $O(|E(H)|)$ time, which is linear in the size of the input hypergraph.

Special case: all-to-all traffic pattern

Wan [50] studied the all-to-all traffic grooming on BLSR networks for minimizing the maximum number of SADMs. For the first step to generate primitive cycles, Wan [50] proposed the following approach to achieve the minimum number of primitive cycles: let n be the number of nodes in the BLSR network, and assume the nodes are labelled from 0 to $n-1$ in the clockwise direction. According to the parity of n there will be the following two cases:

1. If n is even, for every four nodes $i, i+\frac{n}{2}, j, j+\frac{n}{2}$, where $0 \leq i < j \leq \frac{n}{2}$, the following two primitive cycles are constructed:

   - A clockwise primitive cycle formed by traffic demands (i,j),$(j, i+\frac{n}{2})$, $(i+\frac{n}{2}, j+\frac{n}{2})$, $(j+\frac{n}{2}, i)$ routed in the clockwise direction;
   - A counterclockwise primitive cycle formed by traffic demands $(i, j+\frac{n}{2})$,$(j+\frac{n}{2}, i+\frac{n}{2})$, $(i+\frac{n}{2}, j)$, (j,i) routed in the counterclockwise direction.

   For every node i, where $0 \leq i \leq \frac{n}{2}-1$ and $i \bmod 2 \equiv 0$, the following primitive cycle is constructed:

- A clockwise primitive cycle formed by traffic demands $(i, i+\frac{n}{2})$, $(i+\frac{n}{2}, i)$ routed in the clockwise direction.

For every node i, where $0 \leq i \leq \frac{n}{2}-1$ and $i \bmod 2 \equiv 1$, the following primitive cycle is constructed:

- A counterclockwise primitive cycle formed by traffic demands $(i, i+\frac{n}{2})$, $(i+\frac{n}{2}, i)$ routed in the counterclockwise direction.

Therefore the total number of primitive cycles is $2(\frac{\frac{n}{2}}{2})+\frac{n}{2}=\frac{n^2}{4}$, where there are $\lceil\frac{n^2}{8}\rceil$ primitive cycles in one direction and $\lfloor\frac{n^2}{8}\rfloor$ in the other direction.

2. If n is odd, for every four nodes $i, i+\lceil\frac{n}{2}\rceil, j, j+\lceil\frac{n}{2}\rceil$, where $0 \leq i < j < \lfloor\frac{n}{2}\rfloor$, the following two primitive cycles are constructed:

   - A clockwise primitive cycle formed by traffic demands (i,j),$(j, i+\lceil\frac{n}{2}\rceil)$, $(i+\lceil\frac{n}{2}\rceil, j+\lceil\frac{n}{2}\rceil)$, $(j+\lceil\frac{n}{2}\rceil, i)$ routed in the clockwise direction;
   - A counterclockwise primitive cycle formed by traffic demands $(i, j+\lceil\frac{n}{2}\rceil)$, $(j+\lceil\frac{n}{2}\rceil, i+\lceil\frac{n}{2}\rceil)$, $(i+\lceil\frac{n}{2}\rceil, j)$, (j,i) routed in the counterclockwise direction.

   For every node i, where $0 \leq i < \lfloor\frac{n}{2}\rfloor$, the following two primitive cycles are constructed:

   - A clockwise primitive cycle formed by traffic demands $(i, \lfloor\frac{n}{2}\rfloor)$, $(\lfloor\frac{n}{2}\rfloor, i+\lceil\frac{n}{2}\rceil)$, $(i+\lceil\frac{n}{2}\rceil, i)$ routed in the clockwise direction;
   - A counterclockwise primitive cycle formed by traffic demands $(i, i+\lceil\frac{n}{2}\rceil)$, $(i+\lceil\frac{n}{2}\rceil, \lfloor\frac{n}{2}\rfloor)$, $(\lfloor\frac{n}{2}\rfloor, i)$ routed in the counterclockwise direction.

   Therefore the number of primitive cycles is $(\lfloor\frac{\lfloor\frac{n}{2}\rfloor}{2}\rfloor)+\lfloor\frac{n}{2}\rfloor=\frac{n^2-1}{8}$ in each of the clockwise and counterclockwise directions.

Based on the above construction of primitive cycles, we can construct a traffic hypergraph and use algorithm MinMax_Hyper to solve the MMkEPH problem. However, due to the special scheme of constructing primitive cycles, a better approach can be used to compute the solution. We will describe the case for n being even in the following, and the case for n being odd can be solved in a similar way.

If n is even, we assume that $n=2m$ without loss of generality. We construct a complete graph K_m with m nodes as follows: each node i in K_m represents a node pair $(i, i+\frac{n}{2})$ in

the BLSR; the edge between node i and j in K_m represents the two primitive cycles formed by traffic demands (i,j),$(j,i+\frac{n}{2})$, $(i+\frac{n}{2},j+\frac{n}{2})$, $(j+\frac{n}{2},i)$ in the clockwise direction and traffic demands $(i,j+\frac{n}{2})$,$(j+\frac{n}{2},i+\frac{n}{2})$, $(i+\frac{n}{2},j)$, (j,i) in the counterclockwise direction. It is also noticed that there is a primitive cycle between each node pair $(i,i+\frac{n}{2})$, and there are $\frac{m}{2}$ primitive cycles in total for each of the clockwise and counterclockwise directions. We use a *hidden edge* to denote each pair of such primitive cycles in the complete graph K_m. For the complete graph K_m, we can use the same algorithm described in Section 5.2.3 to groom the edges of K_m, and use one extra SADM at each node to groom the hidden edges. Therefore, we have the follow corollary based on Theorem 29.

Corollary 37 *A solution with* $\max_v A(v) \leq \frac{m}{\sqrt{\frac{2}{3}\sqrt{2k}-2}} + \frac{5}{2}$ *can be obtained for the Min-Max grooming of all-to-all traffic in BLSR networks.*

As well, the solution can be improved further to $\frac{N-1}{3}+1 \leq \frac{n}{\sqrt{\frac{3}{4}\sqrt{2k}-3}} + \frac{13}{3}$ for $k \geq 6$, and $\frac{n}{\sqrt{\frac{4}{5}\sqrt{2k}-4}} + \frac{11}{2}$ for $k \geq 10$, and $\frac{n}{\sqrt{\frac{5}{6}\sqrt{2k}-5}} + \frac{38}{5}$ for $k \geq 15$, as derived in Section 5.2.3.

6.2.2 Algorithms for the MaxCkEPH problem

It is observed that Algorithm MaxConnectivity_Grooming proposed in Section 5.3.2 can be directly applied to solve the Maximum Connectivity k-Edge-Partitioning of Hypergraph problem. However, the theoretical analysis of the performance guarantee uses the results of the maximum b-matching of graphs, which are not known for hypergraphs. Therefore, the approximation ratio of Algorithm MaxConnectivity_Grooming can not be preserved for the MaxCkEPH problem.

Special case: all-to-all traffic pattern

As illustrated in Section 6.2.1, we can use a complete graph with some hidden edges to denote the primitive cycles constructed by the approach in [50]. For the Maximum Throughput traffic grooming problem, the algorithm proposed in Section 5.3.3 can be used directly to achieve the same performance. Based on Theorem 35, we have the follow corollary for the Maximum Throughput grooming of all-to-all traffic in BLSR networks.

Corollary 38 *A solution* $\mathcal{E}$ *with* $\sum_{E_i \in \mathcal{E}} |E_i| \geq \frac{mL\lfloor\sqrt{k}\rfloor}{2}$ *can be obtained for the Maximum Throughput grooming of all-to-all traffic in BLSR networks.*

6.3 Empirical results

We implemented Algorithm MinMax_Hyper and conducted simulations to evaluate its practical performance by comparing to a lower bound. In generating traffic hypergraphs, we specified the number n of network nodes and a parameter t that is used to calculate the number n^{1+t} of traffic demands on the BLSR network. Then we used the algorithm *closed chain first* proposed in [51] to generate primitive cycles, which are further formulated as a hypergraph. Figure 6.2 shows the empirical results of Algorithm MinMax_Hyper for $n = 36$ and $t = 0.6$, where for each value of grooming factor k, we collect the results from 100 randomly generated traffic demand sets and compute the average value. The lower bound we used for measuring Algorithm MinMax_Hyper is $\lceil \frac{\Delta(H)}{k} \rceil$, where $\Delta(H)$ represents the maximum degree of hypergraph H. We observe that the experimental performance of the algorithm is not far from the lower bound, and thus the optimum. The gap between the results achieved by the algorithm and the lower bound indicates that there might exist room to improve the performance of the algorithm, or it is possible to show a better lower bound. Simulations are conducted on hypergraphs with different parameter settings as well, and the results present similar characteristics (see Figure 6.3 and Figure 6.4 for other representative results).

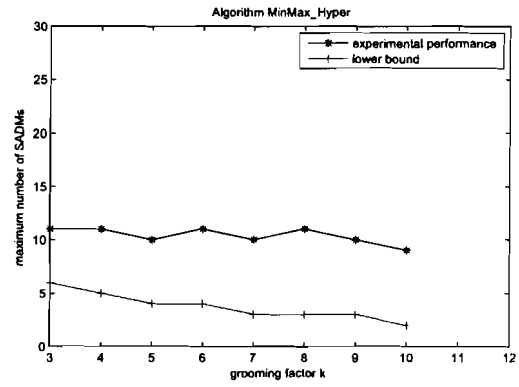

Figure 6.2: Performance of Algorithm MinMax_Hyper for $n = 36$ and $t = 0.6$.

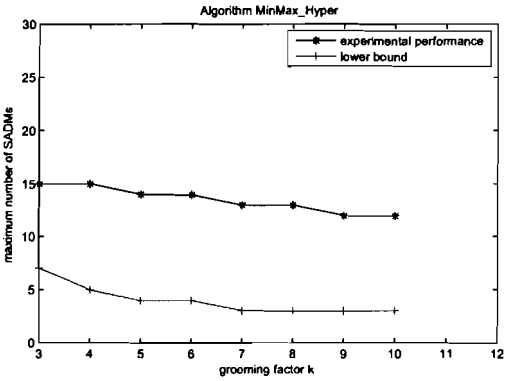

Figure 6.3: Performance of Algorithm MinMax_Hyper for $n = 48$ and $t = 0.6$.

6.4 Summary

In this chapter, we studied the traffic grooming problem in BLSR networks. We extended the graph partitioning approach used for traffic grooming in UPSR networks to the BLSR networks, and formulated the traffic grooming problems into hypergraph partitioning problems. The algorithms proposed in previous chapters for graph partitioning problems are also extended to solve the hypergraph partitioning problems. However, the guaranteed performance can not be preserved as the graph theory results we used in previous chapters are not available for hypergraphs. As a possible future research direction, achieving performance guaranteed algorithms for hypergraph partitioning problems seems interesting yet challenging. Also, the hypergraph partitioning formulation aims to solve the grooming of primitive cycle (i.e., the second step of the two-step approach) rather than the entire traffic grooming problem. As another more difficult problem, achieving performance guaranteed algorithms for the entire traffic grooming problem in BLSR networks is still open. Although the two-step approach provided an efficient way to solve the traffic grooming problem, the two steps are coupled closely and solving each step optimally might not lead to the optimal solution for the entire problem. An integrated approach which considers the two steps jointly is needed to develop performance guaranteed algorithms.

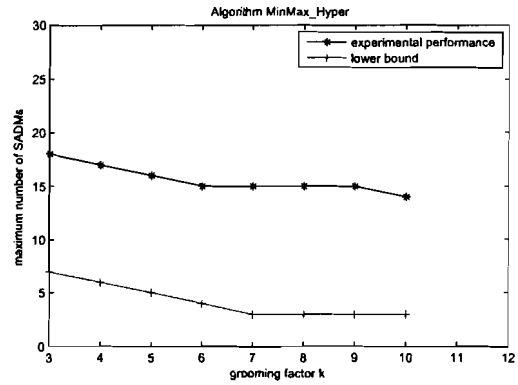

Figure 6.4: Performance of Algorithm MinMax_Hyper for $n = 60$ and $t = 0.6$.

Chapter 7

Discussion for future work

7.1 Traffic grooming in mesh networks

The mesh topology has been increasingly considered recently as an alternative of the ring topology to build the next generation optical networks due to its scalability. The mesh considered in optical networks is an irregular mesh topology. Roughly speaking, an irregular mesh is based on a conventional mesh but there is no restriction that every node must be connected to the four neighbors in the pattern defined in the mesh. In other words, an irregular mesh can be considered as a planar graph with a bounded node degree, in practice the planarity may not be a restriction though. Compared to the regular mesh topology, the irregular mesh topology is more flexible and much easier to scale.

7.1.1 Related work

Due to the increasing deployment of Digital Crossconnects (DXC) in optical networks, the multi-hop traffic grooming problem is usually considered in mesh networks rather than the single-hop traffic grooming problem. Along the path from the source to the destination, each traffic is carried by a *lightpath* for each hop, where the lightpath is defined as a fully optical wavelength circuit spanning multiple physical links. Such a multi-hop traffic grooming problem in mesh networks is usually divided into four sub-problems [67]:

1. Determining the virtual topology that consists of light-paths;

2. Routing the lightpaths over the physical topology;

3. Performing the wavelength assignment to the lightpaths; and

4. Routing the traffic demands on the virtual topology.

Based on the optimization goal, the existing research on traffic grooming in mesh networks can be divided into two categories: the first category is to minimize the total number of SADMs to satisfy a given set of traffic demands, and the second category is to maximize the number of accommodated traffic demands subject to using limited traffic grooming resources. We will discuss the two categories for traffic grooming in mesh network in the following respectively.

Traffic grooming to minimize the total number of SADMs

Similar as the traffic grooming in ring networks, the major goal of traffic grooming in mesh networks is to satisfy a given set of traffic demands such that the total number of used SADMs is minimized. A lot of research papers have proposed heuristic algorithms based on Integer Linear Programming, and the work by Hu and Leida [33] can be considered as a representative one. We will introduce their approach in the following.

Hu and Leida [33] made the assumption that the lightpaths and their routes in the physical topology are given, and only the third and fourth sub-problems mentioned above are considered. They use graph $G_f = (V_f, E)$ to represent the physical topology, where E is the set of physical links and V_f is the set of physical network nodes. They also use graph $G_o = (V_o, L)$ to represent the virtual topology, where L is the set of edges representing lightpaths and $V_o \subset V_f$ is the set of network nodes which are connected via lightpaths. It is noticed that each edge in L actually corresponds to a path in graph G_f. They also assume that each lightpath might contain multiple wavelengths (i.e., each light path is treated as a logical connection between a pair of network nodes). Based on the above definitions, the grooming problem on mesh networks can be described as follows: for a given set of traffic demands, find an optimal way to route and groom these demands in the virtual topology, and also to assign a set of wavelengths to each lightpath such that the total number of used SADMs is minimized. There are two key constraints which should be taken into consideration:

- the wavelength capacity constraint to accommodate low-rate traffic demands, and

- the wavelength continuity constraint for each lightpath (i.e., the same wavelength(s) should be assigned to a lightpath over the physical links it traverses).

Therefore given the above problem setting, the number of SADMs required for each lightpath is equal to twice the number of wavelengths assigned to it (i.e., one SADM is used for one end of each wavelength on a lightpath). Thus, the number of wavelengths on a lightpath can be potentially reduced by grooming multiple low-rate traffic demands onto a single wavelength, and further the number of SADMs can be reduced. Hu and Leida [33] formulate the problem into a Integer Linear Programming (ILP) problem, where the objective is to minimize the number of used SADMs subject to the above two major constraints.

It is well known that the ILP problem is NP-hard, and it may not be computationally feasible to solve the above ILP problem, particularly for large networks. A more efficient approach is proposed to solve the formulated ILP problem in [33]: the ILP problem is decomposed into two smaller sub-problems, where one is traffic grooming and routing (GR) problem and the other is the wavelength assignment (WA) problem. In the GR problem, they only consider how to groom and route demands over lightpaths and ignore the issue of how to assign specific wavelengths to lightpaths, and the size of the GR ILP problem is much smaller than the original ILP problem. Once the GR problem is solved, WA is considered with the goal to derive a feasible wavelength assignment solution. It is worth pointing out that in general the decomposition approach does not yield the optimal solution for the original ILP problem. Therefore, they provided a sufficient condition under which it can be shown that the decomposition approach does produce an optimal solution for the original ILP problem.

Maximum Throughput traffic grooming

Zhu and Mukherjee [67, 66, 65] studied the traffic grooming on mesh networks with different problem settings and optimization goals. They assume that the number of SADMs in each node and/or the number of wavelength available on each optical fiber are/is fixed, and the objective is to establish lightpaths, and groom as many low-rate traffic demands as possible onto the lightpaths (i.e., the objective is to maximize the throughput for a given set of traffic demands).

- ILP based algorithms

Similar as the conventional traffic grooming problem in mesh networks, the Maximum Throughput traffic grooming problem is formulated as an Integer Linear Program (ILP) in [67]. Due to the high computational complexity to solve the ILP problem, two heuristic algorithms are proposed in [67] for large networks, and simulations are conducted to compare the results with the optimal results obtained from the ILP problem. The simulation results suggested that the heuristic algorithms can achieve acceptable performance with relatively low computational complexity.

As well, they extended the problem setting to a network revenue model: different low-rate traffic demands might have different revenues even if they have the same bandwidth requirement. A weight is associated with each low-rate traffic demand to represent the revenue that can be achieved by satisfying the traffic demand. Therefore, the objective becomes to maximize the sum of revenues subject to having fixed number of SADMs and/or fixed number of available wavelengths on each optical fiber. Again, this problem can be formulated as an ILP by simply modifying the ILP used for the Maximum Throughput traffic grooming.

- Algorithms based on a novel generic graph model

  To solve the traffic grooming problem in mesh networks, one approach is to deal with the four sub-problems separately. The ILP decomposition approach in [33] and the heuristic algorithms in [67] fall into this category. Although this divide-and-conquer method makes traffic grooming easier to handle, it cannot guarantee to achieve the optimal solution even if the optimal solution for each sub-problem can be computed. The other approach is to solve the four sub-problems as a whole. The ILP approaches used in [33, 67] fall into this category. By taking into account all the constrains of the four sub-problems simultaneously, the ILP approach is capable of achieving optimal solutions for relatively small networks with reasonable computational complexity. However, the ILP approach is not scalable and cannot be directly applied to large networks.

  Zhu *et al.* [66] proposed a novel graph model, and developed integrated traffic grooming heuristic algorithms based on this model to maximize the throughput for a given set of traffic demands. This graph model is generic in the sense that it can be applied to a wide range of problem settings. For examples, each network node can be equipped with the same number of SADMs or different number of SADMs, each traffic demand

can be a uniform size or non-uniform size. In this graph model, for any given physical network configuration, an auxiliary graph G is constructed. The auxiliary graph G is a layered graph with $(W+2)$ layers, where W is the number of wavelengths on each optical fiber in the network. Layers 1 though W denote the W wavelength layers, layer $(W+1)$ is called the lightpath layer, and layer $(W+2)$ is called the access layer, where a low-rate traffic flow starts and terminates. For each network node, there are two ports on each layer, denoted by two nodes in graph G, an input port and an output port. The edge set in graph G contains the following edges:

- Wavelength bypass edges: there is an edge from the input port to the output port on each wavelength layer at each node;
- Grooming edges: there is an edge from the input port to the output port on access layer at node i if node i has SADMs to perform the grooming;
- Mux edges: there is an edge from the output port on the access layer to the output port on the lightpath layer at each node;
- Demux edges: there is an edge from the input port on the lightpath layer to the input port on the access layer at each node;
- Transmitter edges: there is an edge from the output port on the access layer to the output port on wavelength layer l if there is an SADM operating on wavelength l at a node;
- Receiver edges: there is an edge from the input port on wavelength layer l to the input port on the access layer if there is an SADM operating on wavelength l at a node;
- Converter edges: there is an edge from the input port on wavelength layer l_1 to the output port on wavelength layer l_2 at a node if wavelength l_1 can be converted to wavelength l_2 at the node;
- Wavelength-Link edges: there is an edge from the output port on wavelength layer l at node i to the input port on wavelength layer l at node j if there is a physical link from node i to node j and wavelength l on this link is free;
- Lightpath edges: there is an edge from the output port on the lightpath layer at node i to the input port on the lightpath layer at node j if there is a lightpath from node i to node j. Those lightpath edges are set up with the establishment

of any lightpath during the algorithm, and there is no such edge initially when no lightpath is set up.

Each edge in the auxiliary graph G is associated with a weight, which is used to denote its capacity. For a wavelength-link edge, its capacity is the capacity of the corresponding wavelength on the corresponding link. For a lightpath edge, its capacity is the residual capacity of the corresponding lightpath. For all the other types of edges, the capacity is set to be ∞. It should be clear that the auxiliary graph reflects the current state of the network, and the network can be heterogeneous, with different nodes having different resources and capabilities. Zhu *et al.* [66] used an example to illustrate how to groom a single traffic demand T from source node s to destination node t with a certain bandwidth requirement m: find the shortest path p in graph G from the output port on the access layer of the source node s to the input port on the access layer of the destination node t. If no such path exists, then traffic demand T can not be satisfied given the current available resources. Otherwise if p contains wavelength-link edges, one or more lightpaths going through the corresponding wavelength-links needs to be set up. A lightpath starts whenever p travels through a transmitter edge, follows the subsequent wavelength-link edges, and terminates at the first receiver edge. Route T along the lightpaths in p. If the capacity of the path, which is defined as the minimum capacity of the lightpaths along p, is less than the entire amount m of T, route the maximum amount possible of the traffic (i.e., it is assumed that a traffic demand can be split, and it can be satisfied partially if the entire traffic demand can not be accommodated). After T or a fraction of T is groomed, the graph G is updated accordingly such that the auxiliary graph always reflects the up-to-date network state. The above procedure for grooming a single traffic demand can be naturally extended to groom a given set of traffic demands such that as many traffic demands as possible are satisfied. And the order in which the traffic demands are routed plays an important role in achieving good performance. Several grooming algorithms are proposed in [66] based on different traffic demands selection schemes, and simulations are conducted to show that this integrated grooming approach based on the novel graph model produces good performance by considering the four sub-problems simultaneously.

7.1.2 Future research

Both the traffic grooming problem to minimize the total number of SADMs and the Maximum Throughput traffic grooming problem have been studied in mesh networks. However, no research has been known for the Min-Max traffic grooming problem. Thus studying the computational complexity and developing efficient algorithms for the Min-Max traffic grooming in mesh networks is a possible future research direction.

As we discussed above, a novel graph model has been used to design algorithms for Maximum Throughput traffic grooming in mesh networks. This approach is an integrated one in the sense that it considers all the sub-problems jointly. One interesting open problem is to extend this graph model to tackle the traffic grooming problem to minimize the total number of SADMs, and the Min-Max traffic grooming problem. For the Maximum Throughput traffic grooming problem, the amounts of network resources (e.g., SADMs) are given as input parameters, and thus it is natural to model those parameters as nodes, edges, or weights in the graph. However for the other two optimization goals, the traffic demands are given as the input, and we are supposed to optimize the network resources. Therefore, probably a graph model representing the traffic demands instead of the network resources is more suitable (similar as the traffic graph/hypergraph we defined in the UPSR/BLSR networks). Thus, the difficulty lies in how to construct the graph model, and how to formulate the traffic grooming problem into a problem on the graph model.

As we know, it is NP-hard to solve ILP while Linear Programs (LP) can be computed optimally in polynomial time. For the ILP based algorithms introduced above, one possible research direction is to transfer the ILP problems into LP problems by using techniques such as LP relaxation, and design algorithms to transform the optimal solutions of the LP problems into approximate solutions of the ILP problems. This kind of approach is widely used to design ILP based approximation algorithms for optimization problems. For the traffic grooming problem in mesh networks, the challenge lies in the transform from solutions of the LP problems to the solutions of the ILP problems, and it could be even difficult to obtain feasible solutions of the ILP problems based on the solutions of the LP problems.

7.2 Traffic grooming in other topologies

Besides the ring and mesh topology, traffic grooming in some other fundamental topologies, including path, star, and tree, has also been studied.

7.2.1 Related work

Bermond *et al.* [4] considered the single-hop traffic grooming in path networks, and Huang *et al.* [34] considered the multi-hop traffic grooming in path, star, and tree networks.

Traffic grooming in path networks

- Single-hop traffic grooming

  Bermond *et al.* [4] formulated the single-hop traffic grooming problem in the undirected path as a graph partitioning problem. They used P_N to denote a path with node set $V = \{0, 1, 2, \ldots, N-1\}$ and edge set $E = \{(i, i+1) | 0 \leq i \leq N-2\}$, and I to denote the set of uniform traffic demands, each of which is a pair (u, v) representing the traffic demand from node u to v. In the graph partitioning formulation, traffic demand set I is modelled by a graph $G = (V, E)$ where each edge $e = (u, v)$ is associated to the traffic demand (u, v). Since the physical network topology is a path, each traffic demand is routed uniquely from u to v along P_N, and need to be groomed into a wavelength w. Let $B_w = (V_w, E_w)$ be the sub-graph of G containing all the traffic demands groomed into wavelength w, and $L(B_w, e)$ be the number of traffic demands in B_w which are routed through edge e in P_N, then the traffic grooming problem can be formulated as the following graph partitioning problem:

  Inputs: A path P_N, a grooming factor k, and a set of traffic demands I modelled by the graph $G = (V, E)$;
  Output: A partition of the edges of G into sub-graphs $B_w = (V_w, E_w)$ (where $w = 1, \ldots, W$) such that $L(B_w, e) \leq k$ for each edge e of P_N;
  Objective: Minimize $\sum_{1 \leq w \leq W} |V_w|$.

  Bermond *et al.* [4] optimally solved the above graph partitioning problem for the case that $k = 1$. For $k \geq 2$, the problem is proved to be NP-hard and it is even difficult to develop approximation algorithms for an arbitrary traffic pattern. In particular, when

the grooming factor $k = 2$, the problem is the same as the problem to partition G into the maximum number of triangles. Bermond *et al.* [4] optimally solved the case that $k = 2$ and G is a complete graph (i.e., I is an all-to-all traffic pattern) by using the results of design theory.

- Multi-hop traffic grooming

  Huang *et al.* [34] studied the multi-hop traffic grooming in path networks. The optimization goal is to minimize the total amount of electronic switching over all network nodes. In this cost model, every time a lightpath terminates at a network node, one unit of cost is incurred for each unit traffic stream carried by the lightpath if this stream has to undergo electronic switching (i.e., the stream does not have this node as its destination). This optimization goal indirectly captures the total cost of SADMs. Under this optimization goal, the following traffic grooming problem variants are proved to be NP-hard in [34]:

  - The multi-hop traffic grooming problem in unidirectional path networks is NP-hard.
  - The multi-hop traffic grooming problem in bidirectional path networks is NP-hard.
  - Constant-factor approximation to the optimal solution of the multi-hop traffic grooming problem in unidirectional path networks is NP-hard.

  Since most network topologies contain the path as a sub-topology, the above theorems suggest that it is unrealistic to achieve optimal or constant ratio approximate solutions to the multi-hop traffic grooming problem in general network topologies. A heuristic algorithm for the multi-hop traffic grooming in path networks is proposed in [34], and evaluated by simulation results.

Traffic grooming in star and tree networks

Huang *et al.* [34] studied the multi-hop traffic grooming in star and tree networks as well, where the optimization goal is still to minimize the total amount of electronic switching over all network nodes. They proved that the problem remains NP-hard for both star and tree networks. Heuristic algorithms are proposed for both topologies. In addition, they

derived a series of lower and upper bounds for both star and tree networks, which are increasingly tighter but have considerably higher computational complexity. Those bounds provide a good benchmark to evaluate the performances of the heuristic algorithms proposed for traffic grooming in star and tree networks.

7.2.2 Future research

For the single-hop traffic grooming, previously only the path topology is considered. One possible research direction is to consider the single-hop traffic grooming in star and tree networks as well. This research direction can be conducted in two categories: computational complexity of the problems and efficient algorithms to solve the problems. As to the former, it is noticed that some results on computational complexity of traffic grooming in tree networks might be implied by the known results in path networks, since the path topology is a special case of the tree topology. The similar relation exists between the star and tree topology. For the research on designing efficient algorithms, it is worth exploring both special traffic (e.g., all-to-all) patterns and the arbitrary traffic pattern.

For the multi-hop traffic grooming in path, star, and tree networks discussed above, the optimization goal is to minimize the total amount of electronic switching over all network nodes. This cost model simplifies the traffic grooming problem, however, it is not as accurate as the total number of SADMs in order to represent the cost of the network. Therefore, studying the traffic grooming problem in those fundamental network topologies under the conventional cost model is a possible future research direction, which should include the research on both deriving computational complexity and designing efficient algorithms.

For both the single-hop and multi-hop traffic grooming discussed above, the optimization goal is to minimize the total amount of traffic grooming resources. There is no result known for the Min-Max traffic grooming and Maximum Throughput traffic grooming problems in path, star and tree networks, which can be another possible research direction for traffic grooming on those fundamental topologies.

Chapter 8

Concluding remarks

In this thesis, we studied the traffic grooming problem in WDM optical networks. Traffic grooming is to multiplex low-rate traffic demands into high-speed wavelength channels to make efficient use of the high bandwidth of the wavelength channels in optical networks. Traffic grooming is carried out by SADMs, which are expensive network devices. Therefore, the optimization goal of traffic grooming is usually to either minimize the number (which can be the total number or the maximum number over all network nodes) of required SADMs to satisfy a given set of traffic demands, or maximize the number of satisfied traffic demands subject to using a limited number of SADMs. Those optimization goals have attracted a lot of research attention, however, few results have been known to have guaranteed worst case performance. In this thesis, we focused on developing performance guaranteed algorithms for traffic grooming in UPSR networks. Firstly, we studied the traffic grooming problem to minimize the total number of SADMs. We proposed two linear time approximation algorithms, which achieve better upper bounds on the number of SADMs than previous algorithms. Secondly, we studied traffic grooming of the regular traffic pattern. We proved the NP-hardness, and proposed an approximation algorithm to solve the problem. Thirdly, we studied the Min-Max traffic grooming problem and the Maximum Throughput traffic grooming problem, for each of which we showed the NP-hardness and proposed an approximation algorithm. We also studied the all-to-all traffic pattern, and gave algorithms that achieve solutions constant factors away from the optima. In addition, we studied the traffic grooming problem in BLSR networks, and discussed the possible extensions of the results in UPSR networks to BLSR networks. Finally, we surveyed the research on traffic grooming in mesh networks and other fundamental network topologies, based on which some possible

future research directions are pointed out.

Bibliography

[1] http://www.linktionary.com/m/man.html.

[2] http://www.mathe2.uni-bayreuth.de/markus/reggraphs.html.

[3] A. Aho, J. Hopcroft, and J. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley, Reading, MA, 1974.

[4] J.-C. Bermond, L. Braud, and D. Coudert. Traffic grooming on the path. In *Proceedings of SIROCCO'05*, pages 34–48, 2005.

[5] J.-C. Bermond and S. Ceroi. Minimizing SONET ADMs in unidirectional WDM rings with grooming ratio 3. *Networks*, 41(2):83–86, 2003.

[6] J.-C. Bermond, C.J. Colbourn, A. Ling, and M.-L. Yu. Grooming in unidirectional rings: $k_4 - e$ designs. *Discrete Mathematics*, 284(1-3):57–62, 2004.

[7] J.-C. Bermond and D. Coudert. Traffic grooming in unidirectional WDM ring networks using design theory. In *Proceedings of IEEE ICC'03*, pages 1995–2003, 2003.

[8] J.-C. Bermond, D. Coudert, and X. Muñoz. Traffic grooming in unidirectional WDM ring networks: the all-to-all unitary case. In *Proceedings of The 7th IFIP Working Conference on Optical Network Design and Modelling*, pages 1135–1153, 2003.

[9] R. Berry and E. Modiano. Reducing electronic multiplexing costs in SONET/WDM rings with dynamically changing traffic. *IEEE Journal on Selected Areas in Communications*, 18(10):1961–1971, 2000.

[10] T. Biedl, E.D. Demaine, C.A. Duncan, R. Fleischer, and S.G. Kobourov. Tight bounds on maximal and maximum matchings. *Discrete Mathematics*, 285(1-3):7–15, 2004.

[11] N. Brauner, Y. Crama, G. Finke, P. Lemaire, and C. Wynants. Approximation algorithms for the design of SDH/SONET networks. *RAIRO Operations Research*, 37:235–247, 2003.

[12] B.S. Chen, G.N. Rouskas, and R. Dutta. Traffic grooming in WDM ring networks with the Min-Max objective. In *Proceedings of IFIP NETWORKING'04*, pages 174–185, 2004.

[13] L.W. Chen and E. Modiano. Efficient routing and wavelength assignment for reconfigurable WDM networks with wavelength converters. In *Proceedings of IEEE INFOCOM'03*, pages 1785–1794, 2003.

[14] W. Cho, J. Wang, and B. Mukherjee. Improved approaches for cost-effective traffic grooming in WDM ring networks: uniform-traffic case. *Photonic Network Communications*, 3(3):245–254, 2001.

[15] C. Colbourn and J. Dinitz, editors. *The CRC Handbook of Combinatorial Designs.* CRC Press, 1996.

[16] C. Colbourn and A. Ling. Graph decompositions with application to wavelength add-drop multiplexing for minimizing SONET ADMs. *Discrete Mathematics*, 261:141–156, 2003.

[17] C. Colbourn and P.-J. Wan. Minimizing drop cost for SONET/WDM networks with $\frac{1}{8}$ wavelength requirements. *Networks*, 37(2):107–116, 2001.

[18] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press: Cambridge, Mass. ; McGraw-Hill Book Company: Boston, 1990.

[19] J. Culberson. *Graph coloring page.* http://web.cs.ualberta.ca/ joe/Coloring/.

[20] R. Dutta and G.N. Rouskas. Traffic grooming in WDM networks: past and future. *IEEE Network*, 16(6):46–56, 2002.

[21] M. Fürer and B. Raghavachari. Approximating the minimum degree spanning tree to within one from the optimal degree. In *Proceedings of ACM-SIAM SODA'92*, pages 317–324, 1992.

[22] S. Fiorini and R.J. Wilson. *Edge-colourings of Graphs.* Research Notes in Mathematics, Vol. 16. Pitman, London, 1977.

[23] H.N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of ACM STOC'83*, pages 448–456, 1983.

[24] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and company, NY, 1979.

[25] O. Gerstel, P. Lin, and G. Sasaki. Wavelength assignment in a WDM ring to minimize cost of embedded SONET rings. In *Proceedings of IEEE INFOCOM'98*, pages 94–101, 1998.

[26] O. Gerstel, P. Lin, and G. Sasaki. Combined WDM and SONET network design. In *Proceedings of IEEE INFOCOM'99*, pages 734–743, 1999.

[27] O. Gerstel, R. Ramaswami, and G. Sasaki. Cost effective traffic grooming in WDM rings. In *Proceedings of IEEE INFOCOM'98*, pages 69–77, 1998.

[28] O. Goldschmidt, D.S. Hochbaum, A. Levin, and E.V. Olinick. The SONET edge-partition problem. *Networks*, 41(1):13–23, 2003.

[29] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, Berlin, 1988.

[30] S.L. Hakimi and O. Kariv. On a generalization of edge-coloring in graphs. *Journal of Graph Theory*, 10:139–154, 1986.

[31] I. Holyer. The NP-completeness of some edge-partition problems. *SIAM Journal on Computing*, 10(4):713–717, 1981.

[32] J.Q. Hu. Optimal traffic grooming for wavelength-division-multiplexing rings with all-to-all uniform traffic. *Journal of Optical Networking*, 1(1):32–42, 2002.

[33] J.Q. Hu and B. Leida. Traffic grooming, routing, and wavelength assignment in optical WDM mesh networks. In *Proceedings of IEEE INFOCOM'04*, 2004.

[34] S. Huang, R. Dutta, and G.N. Rouskas. Traffic grooming in path, star, and tree networks: complexity, bounds, and algorithms. *IEEE Journal on Selected Areas in Communications*, 24(4):66–82, 2006.

[35] X. Huang, F. Farahmand, and J.P. Jue. Multicast traffic grooming in wavelength-routed wdm mesh networks using dynamically changing light-trees. *Journal of Lightwave Technology*, 23(10):3178–3187, 2005.

[36] F. Jaeger. A note on sub-Eulerian graphs. *Journal of Graph Theory*, 3:91–93, 1979.

[37] Y. Lee, H.D. Sherali, J. Han, and S. Kim. A branch-and-cut algorithm for solving an intraring synchronous optical network design problem. *Networks*, 35(3):223–232, 2000.

[38] H. Liu and F.A. Tobagi. Traffic grooming in WDM SONET UPSR rings with multiple line speed. In *Proceedings of IEEE INFOCOM'05*, 2005.

[39] H. Liu and F.A. Tobagi. Traffic grooming in WDM/SONET BLSR rings with multiple line speed. In *Proceedings of IEEE GLOBECOM'05*, 2005.

[40] L. Lovász. *Combinatorial problems and exercises.* problem 17(a), chapter 5. North-Holland, 1993.

[41] S. Masuyama and T. Ibaraki. Chain packing in graphs. *Algorithmica*, 6(6):826–829, 1991.

[42] E. Modiano and A. Chiu. Traffic grooming algorithms for reducing electronic multiplexing costs in WDM ring networks. *Journal of Lightwave Technology*, 18(1):2–12, 2000.

[43] E. Modiano and P. Lin. Traffic grooming in WDM networks. *IEEE communications Magazine*, 39(7):124–129, 2001.

[44] J. Peterson. Die theorie der regulären graphen. *Acta Math.*, 15:193–220, 1891.

[45] Rajiv Ramaswami and Kumar N. Sivarajan. *Optical networks - a practical perspective.* Morgan Kaufmann Publishers, San Francisco, CA, 2002.

[46] J. Simmons, E. Goldstein, and A. Saleh. On the value of wavelength-add/drop in WDM rings with uniform traffic. In *Proceedings of OFC'98*, pages 361–362, 1998.

[47] A.K. Somani. Surviable traffic grooming in WDM networks. In *Proceedings of the International Conference on Broad-Band Optical Fibre Communication Technology (BBOFCT'01)*, pages 17–45, 2001.

[48] A. Sutter, F. Vanderbeck, and L. Wolsey. Optimal placement of add/drop multiplexers: heuristic and exact algorithms. *Operations Research*, 46(5):719–728, 1998.

[49] V.G. Vizing. On an estimate of the chromatic class of a p-graph (in russian). *Metody Discret Analiz.*, 3:25–30, 1964.

[50] P.-J. Wan. *Multichannel Optical Networks: Network Theory and Application.* Kluwer Academic, 2000.

[51] P.-J. Wan, G. Calinescu, L. Liu, and Q. Frieder. Grooming of arbitrary traffic in SONET/WDM BLSRs. *IEEE Journal of Selected Areas in Communications*, 18(10):1995–2003, 2000.

[52] P.-J. Wan, L. Liu, and Q. Frieder. Grooming of arbitrary traffic in SONET/WDM rings. In *Proceedings of IEEE GLOBECOM'99*, pages 1012–1016, 1999.

[53] J. Wang, W. Cho, V. Vemuri, and B. Mukherjee. Improved approaches for cost-effective traffic grooming in WDM ring networks: ILP formulations and single-hop and multihop connections. *Journal of Lightwave Technology*, 19(11):1645–1653, 2001.

[54] J. Wang, V. Vemuri, W. Cho, and B. Mukherjee. Improved approaches for cost-effective traffic grooming in WDM ring networks: non-uniform traffic and bidirectional ring. In *Proceedings of IEEE ICC'00*, pages 1295–1299, 2000.

[55] Y. Wang and Q.-P. Gu. Minimizing SONET add-drop multiplexers in optical UPSR networks using minimum wavelengths. *Networks*, page to appear.

[56] Y. Wang and Q.-P. Gu. On the complexity and algorithm of grooming regular traffic in WDM optical networks. *Journal of Parallel and Distributed Computing*, page to appear.

[57] Y. Wang and Q.-P. Gu. Efficient algorithms for traffic grooming in SONET/WDM networks. In *Proceedings of ICPP'06*, 2006.

[58] Y. Wang and Q.-P. Gu. Grooming of symmetric traffic in unidirectional SONET/WDM rings. In *Proceedings of IEEE ICC'06*, 2006.

[59] Y. Wang and Q.-P. Gu. Maximizing throughput for traffic grooming with limited grooming resources. In *Proceedings of IEEE GLOBECOM'07*, 2007.

[60] Y. Wang and Q.-P. Gu. A min-max optimization problem on traffic grooming in WDM optical networks. In *Proceedings of ICCCN'07*, 2007.

[61] D.B. West. *Introduction to graph theory.* Prentice Hall Inc., Upper Saddle River, NJ, 1996.

[62] W. Yao, M. Li, and B. Ramamurthy. Design of sparse grooming networks for transporting dynamic multi-granularity sub-wavelength traffic. In *Proceedings of OFC/NFOEC'05*, 2005.

[63] W. Yao, M. Li, and B. Ramamurthy. Performance analysis of sparse traffic grooming in wdm mesh networks. In *Proceedings of IEEE ICC'05*, 2005.

[64] X. Zhang and C. Qiao. An effective and comprehensive approach for traffic grooming and wavelength assignment in SONET/WDM rings. *IEEE/ACM Transactions on Networking*, 8(5):608–617, 2000.

[65] H. Zhu, H. Zang, K. Zhu, and B. Mukherjee. Dynamic traffic grooming in WDM mesh networks using a novel graph model. *SPIE Optical Networks Magazine*, 4(3):65–75, 2003.

[66] H. Zhu, H. Zang, K. Zhu, and B. Mukherjee. A novel generic graph model for traffic grooming in heterogeneous WDM mesh networks. *IEEE/ACM Transactions on Networking*, 11(2):285–299, 2003.

[67] K. Zhu and B. Mukherjee. Traffic grooming in an optical WDM mesh network. *IEEE Journal on Selected Areas in Communications*, 20(1):122–133, 2002.

[68] K. Zhu and B. Mukherjee. A review of traffic grooming in WDM optical networks: architectures and challenges. *Optical Networks Magazine*, 4(2):55–64, 2003.

[69] K. Zhu, H. Zang, and B. Mukherjee. Design of WDM mesh networks with sparse grooming capability. In *Proceedings of IEEE GLOBECOM'02*, pages 2696–2700, 2002.