Traffic Grooming in Star Networks via Matching Techniques*

Ignasi Sau[1], Mordechai Shalom[2], Shmuel Zaks[1]

[1] Department of Computer Science, Technion, Haifa, Israel, {ignasi,zaks}@cs.technion.ac.il
[2] TelHai Academic College, Upper Galilee, 12210, Israel, cmshalom@telhai.ac.il

Abstract The problem of grooming is central in studies of optical networks. In graph-theoretic terms, it can be viewed as assigning colors to given paths in a graph, so that at most g (the *grooming factor*) paths of the same color can share an edge. Each path uses an ADM at each of its endpoints, and paths of the same color can share an ADM in a common endpoint. There are two sub-models, depending on whether or not paths that have the same color can use more than two edges incident with the same node (*bifurcation allowed* and *bifurcation not allowed*, resp.). The goal is to find a coloring that minimizes the total number of ADMs. In a previous work it was shown that the problem is NP-complete when bifurcation is allowed, even for a star network. In this paper we study the problem for a star network when bifurcation is not allowed. For the case of simple requests - in which only the case of $g = 2$ is of interest - we present a polynomial-time algorithm, and we study the structure of optimal solutions. We also present results for the case of multiple requests and $g = 2$, though the exact complexity of this case remains open. We provide two techniques, which lead to $\frac{4}{3}$-approximation algorithms. Our algorithms reduce the problem of traffic grooming in star networks to several variants of maximum matching problems.

Keywords: Traffic grooming, optical networks, approximation algorithms, maximum matching, Add-Drop Multiplexer.

1 Introduction

1.1 Optical networks

All-optical networks have been largely investigated in recent years due to the promise of data transmission rates several orders of magnitudes higher than current networks [3, 5, 12, 14, 18]. Major applications are in video conferencing, scientific visualization and real-time medical imaging, high-speed supercomputing and distributed computing [7, 12, 18]. The key to high speeds in all-optical networks is to maintain the signal in optical form, thereby avoiding the prohibitive overhead of conversion to and from the electrical form at the intermediate nodes. The high bandwidth of the optical fiber is utilized through *Wavelength-Division Multiplexing (WDM)*: two signals connecting different source-destination pairs may share a link, provided they are transmitted on carriers having different wavelengths (or colors) of light. The optical spectrum being a scarce resource, given communication patterns in different topologies are often designed so as to minimize the total number of used colors, also as a comparison with the trivial lower bound provided by maximum load, that is the maximum number of requests sharing a same physical edge (see [11] for a survey of the main related results).

Though a lot of works have been done in the path and ring topologies, much less is known for more complex networks. The main problem in such networks is the existence of nodes of degree more than two, where traffic can split. Tree networks are thus the first candidates to be studied, and understanding the complexity of the problem for a star network is therefore central step in this direction of finding a general solution for tree networks, and then to general topology networks.

We study the grooming problem (see Section 1.2) in star networks. Despite its simplicity, this topology is important, as it arises in the interconnection of LANs (local area networks) or MANs (metropolitan area networks) with a wide area backbone.

* This research was partially supported by the Israel Science Foundation, grant No. 1249/08.

1.2 The problem

The focus of current studies is to consider the hardware cost. This is modeled by considering the basic switching unit of Add-Drop-Multiplexer (ADM), and focusing on the total number of these ADMs. The key point here is that each lighpath uses two ADMs, one at each endpoint. If two incident lightpaths are assigned the same wavelength, then they can share the ADM at their common endpoint. An ADM may be shared by at most two lightpaths. Moreover, in studying the hardware cost, the issue of *traffic grooming* is central. This problem stems from the fact that the network usually supports traffic that is at rates which are lower than the full wavelength capacity, and therefore the network operator has to be able to put together (= groom) low-capacity requests into the high capacity fibers. At most g requests can share one lightpath. g is termed the *grooming factor*. In terms of ADMs, if g requests of the same wavelength entering through the same edge to one node, they can all use the same ADM at that node, thus saving $g-1$ ADMs. The goal is to find a coloring that minimizes the total number of ADMs. There are two sub-models, depending on whether or not paths that have the same color can use more than two edges touching any node (*bifurcation allowed* and *bifurcation not allowed*, resp.).

In Fig. 1 a star network is presented, with three requests a, b and c. If $g = 2$ then all these requests can get the same color if traffic bifurcation is allowed, and in this case we use a total of 3 ADMs; otherwise, three colors are needed, because any set of two paths will imply a set of three edges touching the center of the star, and in this case we use a total of 6 ADMs. In this work we study the case where bifurcation is not allowed.

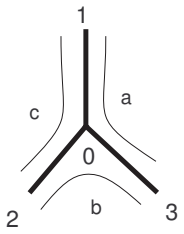

Fig. 1. Star network. Requests a, b, and c can be groomed for $g = 2$ only if traffic bifurcation is allowed.

In graph-theoretic terms, the network is modeled as a graph, each request is viewed as a simple path in the graph, and the traffic grooming problem is viewed as assigning colors to the paths, such that, for any color λ, at most g of the paths colored λ can share the same edge.

For tree networks, and when bifurcation is not allowed, the problem reduces to partitioning the set of requests (paths) into sets (subgraphs) such that in each subgraph (a) the number of paths using any edge is at most g, and (b) the graph induced by the edges of its paths has maximum degree 2. The cost associated with such a subgraph is the number of distinct endpoints of its paths, and the goal is to find a partition that minimizes the sum of the costs of the subgraphs.

1.3 Related works

A review on traffic grooming problems can be found in [20]. In [9] the traffic grooming problem in tree and star networks is studied. The authors extend the approximation results for ring networks of [10] to trees and stars, obtaining an approximation algorithm with approximation factor $2\log g + o(\log g)$, running in polynomial time if g is constant. The traffic grooming model studied in [9] allows bifurcation, and in this case the problem is shown to be NP-complete even for star networks, if $g \geq 3$.

In [13] traffic grooming on path, star, and tree networks is addressed with a deep technological background, but the cost function used to minimize the cost of electronic equipment differs from

that used in [1,2,4,9,10,15,16] among many others. Indeed, in [13] the authors consider the model first stated in [8] (in particular, the function referred as *total amount of electronic switching*), where one unity of cost (namely, a SONET Add-Drop Multiplexer, ADM for short) is incurred each time a wavelength conversion of a request is carried out. That is, what is intended to minimize in this model is the total number of optical hops of all requests from their origins to their destinations. It is assumed that each request requires some electronics at its origin and its destination nodes, regardless of how many requests are terminated at those nodes on each wavelength.

1.4 Summary of results

In this work we deal with the *single hop* problem, where a request is carried along one wavelength, and where bifurcation is not allowed. We study a star network, with $n+1$ nodes $0, 1, 2, \cdots, n$, and a set of edges $\{\{0,1\}, \{0,2\}, \cdots, \{0,n\}\}$. The requests are of length 1 or 2, termed *short* and *long* requests, respectively. We first consider the case of simple requests (Section 3), that is, there are no two requests sharing both endpoints (i.e. identical). In this case only the case $g = 2$ is of interest, and we show a polynomial-time algorithm for the problem. Actually, this result holds also for the case when multiple identical requests of length 2 are allowed. It turns out that a basic component of each solution is a triangle; that is, three requests – one between i and 0, one between j and 0, and one between i and j – all colored with the same color. Indeed, we show that in each optimal solution there is the same number of triangles. We also present (Section 4) results for the case of multiple requests and $g = 2$ (though the exact complexity of this case remains open). We present two techniques, which lead to two $\frac{4}{3}$-approximation algorithms. Our algorithms reduce our problems to the maximum matching problem and several of its variants. We start with preliminaries in Section 2, and conclude with a summary in Section 5.

2 Preliminaries

In this section we provide some preliminaries concerning matchings and the notation we use to denote the subgraphs involved in a partition of the request graph. We use standard graph terminology (cf. for instance [6]). We consider undirected graphs, which may have multiple edges and self-loops.

Matchings. Let $G = (V, E)$ be a (multi)graph with a weight function $w : E \to \mathbb{R}$, and let $\mathbf{I}$ be a function associating an interval of natural numbers with each vertex in V. An $\mathbf{I}$-*matching* is a function $\mathbf{m} : E \to \mathbb{N}$ such that for $v \in V$, $\sum_{e \in E | v \in e} \mathbf{m}(e)$ lies in the interval $\mathbf{I}(v)$. An $\mathbf{I}$-*factor* is an $\mathbf{I}$-matching such that $\mathbf{m} : E \to \{0,1\}$. A *matching* is an $\mathbf{I}$-factor such that $\mathbf{I}(v) = [0,1]$ for each $v \in V$. A *maximum* $\mathbf{I}$-matching is an $\mathbf{I}$-matching $\mathbf{m}$ such that $\sum_{e \in E} \mathbf{m}(e) \cdot w(e)$ is maximized. Maximum $\mathbf{I}$-matchings can be found in polynomial time [17,19]. An $\mathbf{I}$-matching $\mathbf{m}$ corresponds to a sub(multi)graph M of G, such that the multiplicity of the edges of G in M is given by the function $\mathbf{m}$. With slight abuse of notation, M will be also called an $\mathbf{I}$-matching.

Notation. The request graph is denoted R, where a request between two vertices i and j, denoted (i,j), corresponds to an edge of R. The grooming factor is denoted g. A subgraph of R is called *valid* if (a) it respects the grooming constraint, that is, no more than g requests share the same link, and (b) it contains at most two leaves of the star (this is equivalent to forbidding bifurcation). Therefore, the problem we consider can be stated as finding a minimum cost partition of $E(G)$ into valid subgraph, where the cost of a partition is given by the total number of vertices of the subgraphs involved in the partition. When the physical network is a star and the grooming factor is at most 2, the possible subgraphs involved in a partition of $E(R)$ are denoted as follows:

- i denotes the request $(i,0)$.
- $i+j$ denotes the path on three vertices made of the two requests $(i,0)$ and $(j,0)$.
- $i+(i,j)$ denotes the subgraph made of the long request (i,j) and the short request $(i,0)$.
- $[i,j]$ denotes the *triangle* that consists of the three requests $(i,0)$, $(j,0)$, and (i,j).

- $2(i,j)$ (resp. $2i$) denotes the subgraph made of two copies of the request (i,j) (resp. $(i,0)$).
- $2i+j$ denotes the subgraph made of two copies of the request $(i,0)$ and one copy of the request $(j,0)$.
- $2i+2j$ denotes the subgraph made of two copies of the request $(i,0)$ and two copies of the request $(j,0)$.

Note that last four subgraphs corresponding to the three last items are only possible if multiple requests are allowed.

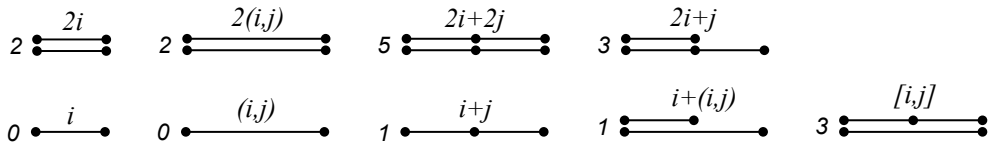

Fig. 2. Possible subgraphs in the star for $g = 2$, together with the notation used to denote them. The number near each subgraph indicates its *savings*. Note that the upper subgraphs are only possible if multiple requests are allowed.

The above subgraphs are shown in Fig. 2, together with the saving in the number of ADMs corresponding to each subgraph. For example, the case $2i+j$ corresponds to a subgraph containing two short requests $(i,0)$ and one short request $(j,0)$. The saving of this subgraph is 3 because if every request stood by its own, they would use a total of 6 ADMS, while by grouping them in a subgraph of type $2i+j$ they use only 3 ADMs, so the saving is of $6-3=3$ ADMs. If only simple requests are given, then we only have the subgraphs depicted in the lower row of Fig. 2, while multiple requests allow also for the subgraphs depicted in the upper row. Note that minimizing the total cost is equivalent to maximizing the total number of savings.

3 Simple requests

We first provide an optimal algorithm in Section 3.1 and then we give a combinatorial characterization of the optimal solutions in Section 3.2.

3.1 Optimal algorithm

Theorem 1. *For any $g \geq 1$, there exists a polynomial time polynomial algorithm for* Star Traffic Grooming *problem for the case of simple requests.*

Proof. Let first $g = 1$. Each long request (i,j) uses 2 ADMs, and no saving is possible. Therefore, only short requests of type i can be matched and save 1 ADM at the central node 0, so if the instance contains x short requests, $\lfloor \frac{x}{2} \rfloor$ savings can be done, and such a solution is optimal.

If grooming is allowed, but no multiple requests are allowed, then note that the case $g = 2$ is the only interesting case, since for arbitrary $g \geq 2$, as we do not allow bifurcations, the only possible subgraphs are those depicted in the lower row of Fig. 2. So, we focus henceforth on the case $g = 2$.

We claim that the following Algorithm SimpleMatch

Algorithm SIMPLEMATCH:
Input: A star S and a set R of *simple* requests between pairs of vertices in S.
Output: a partition of $E(R)$ into a set of valid subgraphs for $g = 2$.

(1) Build an edge-weighted graph $G = (V, E)$ as follows:
  (1.1) For each short request $(i, 0)$ in R, add a vertex v_i to V.
  (1.2) For each long request (i, j) in R, add a vertex v_{ij} to V.
  (1.3) For each vertex $v_{ij} \in V$, if the vertex v_i (resp. v_j) is in V, add the edge $\{v_{ij}, v_i\}$ (resp. $\{v_{ij}, v_j\}$) to E with weight 1.
  (1.4) For each pair of vertices $v_i, v_j \in V$, add the edge $\{v_i, v_j\}$ to E with weight 3 if the vertex v_{ij} is in V, and add it with weight 1 otherwise.
(2) Find a maximum weighted matching M in G.
(3) Output the set of subgraphs corresponding to the edges in M. If some request is left, output it as a subgraph itself.

returns an optimal solution in polynomial time.

The possible subgraphs involved in the partition of the request set are the five lower subgraphs of Fig. 2. We only need to show that all the possible subgraphs, together with their savings, correspond to the weighted edges of the graph G build in the description of the algorithm.

The subgraphs of type $i + (i, j)$ are captured by the edges $\{v_{ij}, v_i\}$, and their weight does correspond to the unitary savings. Also the subgraphs of type $i + j$ when the request $(i, j) \notin R$ are captured by the edge $\{v_i, v_j\}$ with weight 1. Finally, let us see how triangles are represented in G. The key idea is the following.

Claim 2 *If for two integers i, j the three requests $(i, 0)$, $(j, 0)$, and (i, j) are in R, there exists an optimal solution not containing the subgraph $i + j$.*

Indeed, assume that an optimal solution with cost OPT contains the subgraph $i + j$. We can remove the request (i, j) from wherever it is, add it to the subgraph $i + j$, and then the obtained solution containing the triangle $[i, j]$ has cost $OPT' \leq OPT$.

Due to Claim 2, whenever three requests $(i, 0)$, $(j, 0)$, and (i, j) belong to R, we do not need to consider the subgraph $i + j$. In this case, the edge $\{v_i, v_j\}$ with weight 3 corresponds to the triangle $[i, j]$. Note that if the edge $\{v_i, v_j\}$ is in the matching M, it prevents the vertex v_{ij} to be used again, since the only neighbors of v_{ij} in G are v_i and v_j.

Thus, a maximum weighted matching in G (plus possibly, some single requests) corresponds to a valid solution for the instance R maximizing the total savings, or equivalently minimizing the total number of ADMs.

In fact, when the requests are simple, for any $g \geq 3$ the problem is also solved by Algorithm SIMPLEMATCH. This is because, as we mentioned above, due to the model that we are assuming the set of subgraphs involved in any partition of the set of requests is the same for any $g \geq 2$. □

3.2 Structure of an optimal solution

Assume that the maximal number of triangles in a given instance of the problem is k. We show now that *all* optimal solutions contain k triangles. Note that even if Algorithm SIMPLEMATCH does not take into account this property, Lemma 1 provides structural information about the optimal solutions, which is of interest by itself.

We are given a set of requests $R = \{\ell_1, \cdots, \ell_x, s_1, \cdots, s_y\}$, where $\ell_1, \cdots, \ell_x$ are requests of length 2 and $s_1, \cdots, s_y$ are requests of length 1. The maximal number of triangles in R is denoted by $\max(R)$.

Lemma 1. *Let R be a given set of requests on a star network, and let* $\max(R) = k \geq 0$*, let* $MAX = \{t_1 = [1,2], t_2 = [3,4], \cdots, t_k = [2k-1, 2k]\}$ *be a maximal set of triangles in R, and let OPT be any optimal solution for R. Then OPT contains exactly k triangles.*

Proof. Assume that OPT does not contain exactly $0 < m \leq k$ triangles of MAX (so, OPT contains $k - m$ of the triangles in MAX) . W.l.o.g. assume these are the triangles $t_1, t_2, \cdots, t_m$. We define a function f that will assign to each triangle in $t_1, t_2, \cdots, t_m$ a triangle or a pair of triangles in $OPT - MAX$.

Consider any of these triangles $t_j = (2j-1, 2j)$, $1 \leq j \leq m$.

Since $t_j \notin OPT$, we have to consider two cases, according to whether the request $(2j-1, 2j)$ is paired in OPT with one short request (Case a) or it is by itself in OPT (Case b):
Case a: $(2j-1, 2j)$ is paired in OPT with $(2j-1, 0)$ (or, similarly, with $(2j, 0)$).
Consider the request $(2j, 0)$. If $(2j, 0)$ is paired with one short request, or with one long request, or it is by itself in OPT, then by moving $(2j, 0)$ to join $(2j-1, 2j)$ and $(2j-1, 0)$ we get a solution SOL with $cost(SOL) < cost(OPT)$, a contradiction. So we conclude that in this case $(2j, 0)$ is in another triangle t (which is clearly neither of the triangles $t_1, t_2, \cdots, t_m$). In this case we define $f(t_j) = t$.
Case b: $(2j-1, 2j)$ is a component containing only itself in OPT.
Consider the request $(2j-1, 0)$ and $(2j, 0)$. They cannot form together one component of OPT, they cannot form two components with two other short requests, and neither of them can be a component of itself in OPT; this is since in each of these cases we could add them to $(2j-1, 2j)$ and get a solution SOL with $cost(SOL) < cost(OPT)$, a contradiction. Hence, they are matched to long requests in OPT. They cannot both be paired to long requests alone, since then a solution SOL that will add them to $(2j-1, 2j)$ will satisfy $cost(SOL) < cost(OPT)$, a contradiction. So, at least one of them is in a triangle t of OPT. In this case we define $f(t_j) = t$. If both of them are connected to triangles t_j^1 and t_j^2 then we define $f(t_j) = \{t_j^1, t_j^2\}$.

At this point, the set of triangles in $t_1, t_2, \cdots, t_m$ is partitioned into two subsets T_1 and T_2 of sizes s_1 and S_2, respectively, where $s_1 + s_2 = m$. Each triangle in T_1 is mapped to one triangle in OPT. We claim that this mapping is 1-1. Otherwise there are two triangles t_j and $t_{j'}$ mapped to the same triangle of OPT. This is because w.l.o.g. OPT contains the components $2j - 1 + (2j-1, 2j)$ and $2j' - 1 + (2j'-1, 2j')$ and the triangle $[2j, 2j']$; these components contribute 9 ADMs to the solution. The solution OPT' obtained from OPT by taking out all these components and adding $t_j, t_{j'}$ and $(2j, 2j')$ uses one less ADMs, a contradiction. These triangles correspond to s_1 triangles of OPT.

Each triangle in T_2 is mapped to two triangles in OPT. These triangles in OPT contain the short requests $(2j-1, 0)$ and $(2j, 0)$. The number of vertices > 0 in these triangles is thus at least s_2, and therefore that $\cup_{t \in T_2} f(t)$ contain at least s_2 triangles. Since it must contain at most s_2 triangles, it follows that the number of these triangles is exactly s_2, and thus the total number of triangles in OPT in $\cup_{t \in T_1 \cup T_2} f(t)$ is m. These, together with the $k - m$ triangles in OPT that belong to MAX, sum up to a total of k triangles in OPT. This completes the proof. □

4 Multiple requests

4.1 Motivation

If multiple requests are allowed, the problem becomes more complicated. Besides the subgraphs of type $2(i, j)$, the greedy removal of any type of subgraph will not lead to an optimal solution. First we show an example where the pairing if identical short requests leads to a sub-optimal solution: Consider a star with 3 leaves a, b, and c with the request set $R = \{a, a, b, c, (a, b), (a, c)\}$. If we pair the two short requests a in the same component, the cost will be at least 8 whereas an optimal solution uses two triangles $[a, b]$ and $[a, c]$. One could be tempted to remove greedily all the triangles, or similarly a set of triangles of maximum cardinality. The following is a counter example showing that

this strategy is not optimal. Consider the request set $R = \{a, a, b, b, (a, b)\}$. A solution containing the only triangle $[a, b]$ will have a cost of at least 6 ADMs, as opposed to the cost of 5 ADMs in the optimal solution $(a, b), 2a + 2b$.

First we provide in Subsection 4.2 an approximation algorithm in the same spirit of Section 3. Namely, we construct an auxiliary edge-weighted graph G made of appropriate gadgets that capture the possible subgraphs involved in a partition of the request graph, and then we find a feasible solution to our problem by optimally solving a maximum **I**-matching problem in G. We then present another approach in Subsection 4.3, also based on **I**-matching. We prove that these two apparently different algorithms constitute a 4/3-approximation to the problem for $g = 2$. Before presenting the algorithms, we make an observation to be used in the sequel.

Claim 3 *The subgraphs of type $2(i, j)$ can be greedily removed from R without changing the cost of an optimal solution.*

Proof. Assume that R contains two copies of the request (i, j), and that in an optimal solution OPT these copies belong to different subgraphs B_1 and B_2. Since $|V(B_1)| \geq 2$ and $|V(B_2)| \geq 2$, the solution OPT' obtained from OPT by replacing the subgraphs B_1 and B_2 with $B'_1 = (B_1 \setminus (i, j)) \cup (B_1 \setminus (i, j))$ and $B'_2 = 2(i, j)$ satisfies $cost(OPT') \leq cost(OPT)$. □

By Claim 3, we assume henceforth that the requests of type (i, j) (i.e., long requests) are simple. For any leaf i of a star S, denote by s_i (resp. ℓ_i) the number of *short* (resp. *long*) requests of node i in the request set R. We also let $S = \sum_i s_i$ and $L = \sum_i \ell_i$.

4.2 First approach

Our first approach is described in Algorithm MultipleMatch1.

Algorithm MultipleMatch1:
Input: A star S and a set R of (possibly *multiple*) requests between pairs of vertices in S.
Output: a partition of $E(R)$ into a set of valid subgraphs for $g = 2$.

(1) Build an auxiliary edge-weighted multigraph $G = (V, E)$ as follows:
  (1.1) For each leaf i in S, add a new vertex v_i to V.
  (1.2) For each vertex $v_i \in V$, add a self-loop in v_i to E with weight 2.
  (1.3) For each long request (i, j) in R, add two new vertices v_{ij}, v'_{ij} to V and the following edges to E: $\{v_i, v_{ij}\}$ with weight 1, $\{v_j, v'_{ij}\}$ with weight 1, $\{v_{ij}, v'_{ij}\}$ with weight 1, and $\{v_{ij}, v'_{ij}\}$ with weight -1.
(2) Define the following function **I**, which associates an interval of natural numbers with each vertex in V:
  (2.1) $\mathbf{I}(v_i) = [0, s_i]$ for each $v_i \in V$.
  (2.2) $\mathbf{I}(v_{ij}) = \mathbf{I}(v'_{ij}) = [2, 3]$ for each $v_{ij}, v'_{ij} \in V$.
(3) Find a maximum **I**-matching M of G using the algorithm of [17].
(4) Output the following subgraphs of R according to M:
  (4.1) For each self-loop of vertex v_i in M, output subgraph $2i$.
  (4.2) For each i, j in S, if the two edges $\{v_i, v_{ij}\}, \{v_j, v'_{ij}\}$ and the edge $\{v_{ij}, v'_{ij}\}$ with weight 1 are in M, output subgraph $[i, j]$.
  (4.3) For each i, j in S, if the edge $\{v_i, v_{ij}\}$ and the two copies of the edge $\{v_{ij}, v'_{ij}\}$ are in M, output subgraph $i + (i, j)$.
  (4.4) For each i, j in S, if the edge $\{v_j, v'_{ij}\}$ and the two copies of the edge $\{v_{ij}, v'_{ij}\}$ are in M, output subgraph $j + (i, j)$.
  (4.5) If some request is left, output it as a subgraph itself.

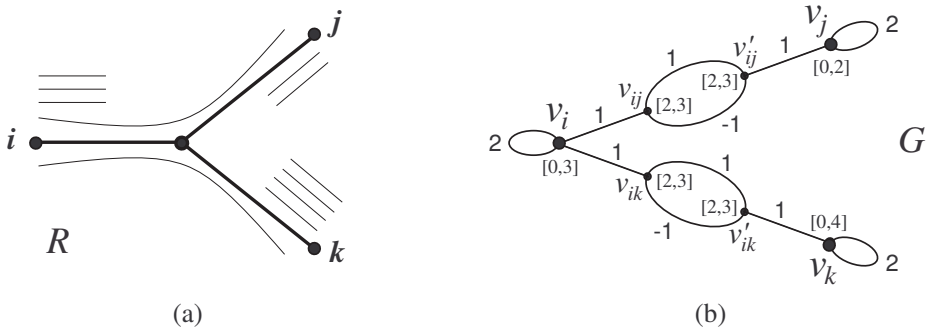

Fig. 3. (a) A traffic instance I in a star on three vertices i, j, k, with $s_i = 3$, $s_j = 2$, and $s_k = 4$. (b) Auxiliary graph G and function $\mathbf{I}$ built in Algorithm MULTIPLEMATCH1. The number beside each edge indicates its weight, while the interval in brackets beside each vertex indicates its allowed degrees.

An example of the graph G and the function $\mathbf{I}$ built in Algorithm MULTIPLEMATCH1 is illustrated in Fig. 3 for a simple star on four vertices. We now briefly discuss the intuition behind the algorithm. For each leaf i of S, the degree bounds at node v_i assure that no more than s_i short requests can be used at vertex i. The degree bounds at vertices v_{ij}, v'_{ij} make sure that the output of the $\mathbf{I}$-matching algorithm can be indeed translated to a partition of the requests in R, and such the savings of each subgraph correspond to the sum of the weights of the edges in M corresponding to this subgraph. As one can check from step (4) of Algorithm MULTIPLEMATCH1, we do not use any subgraph of type $2i + 2j$, $2i + j$, or $i + j$. The key point is that the gadgets used by the algorithm capture simultaneously all other possible subgraphs. The fact of *forgetting* some subgraphs has of course direct consequences in the worst-case performance of the algorithm, as stated in the following theorem.

Theorem 4. *Algorithm* MULTIPLEMATCH1 *is a polynomial-time* 4/3*-approximation algorithm for the* STAR TRAFFIC GROOMING *problem for* $g = 2$ *when multiple requests are allowed.*

Proof. First, the algorithm clearly runs in polynomial time, as the algorithm of [17] used in step (3) does so. We now argue that the output of the algorithm defines a feasible solution to STAR TRAFFIC GROOMING for $g = 2$. From steps (4.1)-(4.5) it follows that the number of short requests used at each vertex i of S by the output of Algorithm MULTIPLEMATCH1 is equal to the degree (taking into account the multiplicity of the edges, and considering that a self-loop induces degree 2) of v_i in the $\mathbf{I}$-matching M of G. Since by definition of $\mathbf{I}(v)$, the degree of v_i in M is at most s_i, no more than s_i short requests are used at vertex i.

In order to analyze the approximation ratio of the algorithm, we now discuss how the output M reflects the cost of the solution of STAR TRAFFIC GROOMING that it defines. (Recall Fig. 2 for the definition of the possible subgraphs and their associated savings.) In case (4.1), for each vertex i in S, a subgraph of type $2i$ has an associated saving of 2, which corresponds to the weight of a self-loop at vertex v_i in G. Let i, j be two vertices in S such that the long request (i, j) is in R. In case (4.2), if the two edges $\{v_i, v_{ij}\}, \{v_j, v'_{ij}\}$ and one copy of $\{v_{ij}, v'_{ij}\}$ belong to M, then the fact that M is a maximum $\mathbf{I}$-matching implies that the copy of $\{v_{ij}, v'_{ij}\}$ in M is the one with positive weight. In this case, the sum of the weights of these three edges is 3, which is equal to the saving of a triangle $[i, j]$. In case (4.3), the edge $\{v_i, v_{ij}\}$ and the two copies of the edge $\{v_{ij}, v'_{ij}\}$ are in M, so the sum of the weights of these three edges is 1, which corresponds to the saving of the subgraph $i + (i, j)$. Case (4.4) is symmetric to case (4.3). If none of cases (4.2), (4.3), or (4.4) holds, then none of the edges $\{v_i, v_{ij}\}$ and $\{v_j, v'_{ij}\}$ is in M. From the degree constraints at vertices v_{ij}, v'_{ij}, one can check that the only remaining feasible possibility for an $\mathbf{I}$-matching is to take both copies of the edge $\{v_{ij}, v'_{ij}\}$, therefore incurring a total weight of 0, which indeed corresponds to not saving any ADM.

Finally, in case (4.5), a subgraph of R made of a single request has a saving of 0.

From the above discussion, it follows that there is a bijective correspondence between the gadgets used by the **I**-matching and subgraphs of type $2i$, $[i,j]$, or $i+(i,j)$. Therefore, our algorithm finds the best solution *under the constraint* of not using the other subgraphs that incur some saving, namely those of type $2i+2j$, $2i+j$, or $i+j$ (see Fig. 2).

Given an instance R, let OPT be the cost of an optimal solution S^*, and let ALG be the cost given by Algorithm MultipleMatch1. We construct from S^* another solution S with cost C as follows. For every subgraph B of type $2i+2j$, $2i+j$, or $i+j$ in S^*, we split B into two subgraphs B_i and B_j, where B_i (resp. B_j) contains the short requests of vertex i (resp. j). Note that for each such subgraph B, the cost of B in S^* is 3, while the cost of B_i plus the cost of B_j in S is 4. As all the other subgraphs remain unchanged, it follows that $C \leq \frac{4}{3} \cdot OPT$. But as no subgraph of type $2i+2j$, $2i+j$, or $i+j$ is in solution S, the solution found by Algorithm MultipleMatch1 is equal or better than S, so $ALG \leq C$, which in turn implies that $ALG \leq \frac{4}{3} \cdot OPT$. □

4.3 Second approach

In this section we propose an algorithm that uses an oracle (called TriangleOracle in the algorithm) providing part of the subgraphs, namely all the triangles $[i,j]$ of the solution. We show that the algorithm is optimal provided that the oracle is optimal. In this way we reduce the problem to the problem of finding an optimal set of triangles.

Algorithm MultipleMatch2:
Input: A star S and a set R of (possibly *multiple*) requests between pairs of vertices in S.
Output: a partition of $E(R)$ into a set of valid subgraphs for $g=2$.

(1) Build an unweighted graph $G=(V,E)$ as follows:
  (1.1) V is the set of leaves of S, that is $V=\{1,\ldots,n\}$.
  (1.2) $\{i,j\} \in E$ whenever the request (i,j) belongs to R.
 // *Note that $d_G(i)=\ell_i$ for each $i=1,\ldots,n$.*
(2) Invoke the algorithm TriangleOracle$(G,\mathbf{s})$, where $\mathbf{s}$ is the vector of the values s_i of the number of short requests i. The algorithm returns a subgraph T of G.
(3) For each edge $\{i,j\} \in E(T)$, return the triangle $[i,j]$ as a subgraph and remove it from R.
(4) For each node $i \in V(G)$ build $\lfloor \frac{s_i - d_T(i)}{2} \rfloor$ subgraphs of type $2i$ and at most one subgraph of type i (but do not remove them from R).
(5) For each subgraph of type i built in the previous step, choose arbitrarily a request $(i,j) \in R$, build the subgraph $i+(i,j)$ and remove it from R. If no such request exists, do nothing.
(6) Build a complete n-partite graph whose nodes are the subgraphs built at step (4) that are still in R. There is an edge between two subgraphs if they correspond to different nodes of S. Calculate a maximum matching of this graph. Each edge of this matching corresponds to a subgraph of the form $2i+2j$, $2i+j$ or $i+j$, and each unmatched node of corresponds to a subgraph of type i. Return all these subgraphs.

From the above description of the algorithm, it follows that the set of subgraphs of type $[i,j]$ (i.e. triangles) returned by algorithm MultipleMatch2 corresponds to the edges $E(T)$ of the subgraph returned by TriangleOracle. For this reason, in the rest of this section we will use the terms *edge* and *triangle* interchangeably. In order algorithm MultipleMatch2 to be optimal, it is a necessary condition that TriangleOracle returns the set of triangles of an optimal solution. We will prove that this is also a sufficient condition.

Theorem 5. *If the set of triangles $E(T)$ returned by* TriangleOracle *is the set of triangles of some optimal solution, then* MultipleMatch2 *returns an optimal solution.*

Proof. We have to show that the decisions taken at steps (4), (5) and (6) are correct. We start with step (4).

We say that a request i *paired* if its subgraph contains another request i, i.e., i is in a subgraph of type $2i + 2j$, $2i + j$, or $2i$, and *unpaired* otherwise. We claim that there is an optimal solution that does not contain two subgraphs G_1 and G_2 such that both contain an unpaired request i. This implies that the decision taken by the algorithm in step (4) is optimal.

Indeed, consider an optimal solution with the same set of triangles returned by our algorithm, i.e. the set $E(T)$. Assume by contradiction that it contains two such subgraphs G_1 and G_2. If one of these subgraphs (without loss of generality G_1) does not contain a long request (i, j), then the request i in G_2 can be moved to the subgraph G_1. In this case it does not increase the cost of G_1, because it shares the ADMs of the request i in G_1, and its removal from G_2 does not increase the cost of G_2. Otherwise both G_1 and G_2 contain long requests (i, j_1) and (i, j_2) respectively. Moreover, $j_1 \neq j_2$ because by Claim 3 we assume that the long requests form a simple set of requests. Also neither one of G_1 and G_2 is a triangle, because these triangles are not returned by our algorithm, and therefore are not in this optimal solution. Therefore $G_1 = i + (i, j_1)$ and $G_2 = i + (i, j_2)$ and they use 6 ADMs in total. In this case we can replace G_1 and G_2 with the three subgraphs $2i$, (i, j_1), and (i, j_2) having a total cost of 6 ADMs, to obtain an optimal solution as claimed.

We proceed with the correctness of step (5). In the rest of the proof we consider the cost of a solution in the following way. The L long requests incur a fixed cost of $2L$ ADMs. All other (i.e. short) requests sharing one of these ADMs do not incur any cost for these ADMs. The first observation is that, if there is an unpaired request i at the beginning of this step, a subgraph of type $i + (i, j)$ is an optimal subgraph for it. This is because it incurs a cost of at most 1 ADM (at node 0) in this case, and in any other case it will incur a cost of at least 1 ADM (at node i). The second observation is that no two unpaired requests i and j and a long request (i, j) can exist in R at this point. This is because in this case they would incur a total cost of 1 in the triangle $[i, j]$, and in any other subgraph each one of them incurs a cost of 1, contradicting the optimality of the triangles returned by TriangleOracle. In other words, no two short requests i and j can "compete" for a long request (i, j), therefore greedily forming the subgraphs $i + (i, j)$ will not cause a conflict.

The correctness of step (6) is now almost straightforward. At this point we are left with subgraphs of type i and $2i$ which might be merged to form bigger subgraphs. It can be checked that all the other subgraphs can not be merged. Moreover, these subgraphs can be merged only in pairs, namely pairs of the form $i + j$, $2i + j$, or $2i + 2j$. Each such merging operation reduces the cost of the solution by one ADM. Therefore the goal of the algorithm is to maximize the number of these merging operations. This is equivalent to calculate the maximum cardinality matching of the auxiliary graph built in step (6). □

Following the above result, our goal is to find an algorithm TriangleOracle(G, s) that returns the set of triangles of some optimal solution. Having in mind the counter examples presented at the beginning of this section, we present the following lemma that gives a partial characterization of an optimal set of triangles.

Lemma 2. *There is an optimal solution S^* with the following property: Let T be the set of triangles of S^* and let $[i, j] \in T$. Then either $s_i - d_T(i)$ is even or S^* contains a subgraph $i + (i, j)$.*

Proof. We consider the optimal solution S^* that is returned by the algorithm using the set of triangles T returned by the oracle. Let us consider also a triangle $[i, j] \in T$. Assume that $s_i - d_T(i)$ is odd and there is no subgraph $i + (i, j)$ in S^*. Then after step (3) there will be an odd number of i requests left in R. Therefore there will be one unpaired request i after step (4). This request will not participate in a subgraph $i + (i, j)$ at step (5) by the assumption. Therefore it will remain unpaired in S^*. In this case we can obtain a solution S^{**} from S^* by removing the request i from the triangle $[i, j]$, and moving it to the subgraph containing this unpaired request without increasing the cost. If j has also this property that this leads to a contradiction to the optimality of S^*, otherwise S^{**} is the claimed optimal solution. □

By the above Lemma we can restrict ourselves to algorithms not returning any triangle $[i, j]$ in T if this causes $s_i - d_T(i)$ (or $s_j - d_T(j)$) to be odd. For this reason we propose the following algorithm as a first attempt towards a TriangleOracle.

Algorithm TrianglesViaIFactor:
Input: A Graph $G = (V, E)$, and a vector $\mathbf{s}$ of numbers indexed by V.
Output: A subgraph T of G.

(1) Define the function $\mathbf{f} : V \to \mathbb{N}$ as follows:

$$\mathbf{f}(i) = \begin{cases} s_i, & \text{if } s_i \leq \ell_i \\ \ell_i - (s_i - \ell_i) \bmod 2, & \text{otherwise} \end{cases}$$

(2) Find a maximum $\mathbf{I}$-factor T in G, where $\mathbf{I}(i) = [0, \mathbf{f}(i)]$ for each $i \in V$.

Theorem 6. MultipleMatch2 *is a* 4/3*-approximation for the* Star Traffic Grooming *problem if it uses* TrianglesViaIFactor *as* TriangleOracle.

Proof. Let S be a solution returned by the algorithm and let S^* be an optimal solution. Let T' be the set of triangles of $S \setminus S^*$. Consider the $2|T'|$ short requests in these triangles (of S). Let x be the number of short requests participating in the same subgraphs as these requests in S^*. Then the cost OPT of S^* satisfies

$$OPT \;\geq\; \frac{3}{4}(2|T'| + x) = \frac{3}{2}|T'| + \frac{3}{4}x$$

because these subgraphs are not triangles by the way T' is chosen, and in any subgraph of another type a short request incurs a cost of 3/4 in average, the best case being a subgraph of type $2i + 2j$ using 3 ADMs for 3 short requests.

On the other hand, the cost ALG of S satisfies

$$ALG \;\leq\; |T'| + x \leq \frac{3}{2}|T'| + x$$

because all these x requests are either paired in S, or part of a subgraph of type $i + (i, j)$. In both cases each such request incurs a cost of at most 1. Comparing the right hand sides of the above inequalities we conclude the claim. □

However the proposed algorithm is not optimal, as the following lemma shows:

Lemma 3. *There is an instance for which* MultipleMatch2 *using* TrianglesViaIFactor *as* TriangleOracle *returns a sub-optimal solution.*

Proof. Consider the following instance, on a star with 4 leaves, with $R = \{(1, 2), (2, 3), (3, 4), (4, 1), 1, 2, 2, 2, 2, 4, 4, 4, 4, 4\}$.

In this case G is a C_4, $\ell = (1, 1, 1, 1)$, $\mathbf{s} = (1, 4, 0, 5)$ the $\mathbf{f} = (1, 2, 0, 1)$, and any maximum $\mathbf{I}$-factor has cardinality 1. The oracle might return the singleton $T = \{(1, 2)\}$ which leads to a solution with a cost of 16 ADMs. On the other hand, there is a solution with $T^* = \{(4, 1)\}$ as the set of its triangles, implying a cost 15 ADMs. □

5 Conclusions

We studied the traffic grooming problem in star networks when bifurcation is not allowed. We presented a polynomial-time algorithm for the case of simple requests, and gave some insight into the structure of an optimal solution. Though the algorithm can be extended to the case of multiple long requests, the status of the problem when multiple short requests are allowed remains unsolved.

We presented two approaches with good approximation guarantee using matching techniques. We expect our techniques to lead to a polynomial-time algorithm for the case $g = 2$. For instance, in the approach we presented in Section 4.2, in order to obtain a polynomial-time algorithm for the problem it would be enough to find the right gadgets that also capture the *missing* subgraphs. In fact, the only subgraph we have not been able to capture is the *quadruple* given by two pairs of short requests, so we suspect that either such a gadget may be found, or the quadruple would probably play a distinguished role in a possible NP-completeness proof.

While, according to our results, matching techniques prove to be very helpful for the case $g = 2$, it is not clear how to use them for $g > 2$. It might well be the case that more complicated techniques are needed to deal with higher values of the grooming factor, even for this apparently simple network topology. We believe that our study sheds light on the complexity of traffic grooming for networks whose maximal degree is more than two. Among them, it will be of interest to study the complexity of the problem for tree networks, bounded degree networks, or planar networks.

References

1. O. Amini, S. Pérennes, and I. Sau. Hardness and Approximation of Traffic Grooming. *Theoretical Computer Science*, 410(38-40):3751–3760, 2009.
2. J.-C. Bermond, L. Braud, and D. Coudert. Traffic grooming on the path. *Theoretical Computer Science*, 384(2-3):139–151, 2007.
3. C. A. Brackett. Dense wavelength division multiplexing networks: principles and applications. *IEEE Journal on Selected Areas in Communications*, 8:948–964, 1990.
4. T. Chow and P. Lin. The ring grooming problem. *Networks*, 44(3):194–202, 2004.
5. N. K. Chung, K. Nosu, and G. Winzer. Special issue on dense wdm networks. *IEEE Journal on Selected Areas in Communications*, 8, 1990.
6. R. Diestel. *Graph Theory*, volume 173. Springer-Verlag, 2005.
7. D. H. C. Du and R. J. Vetter. Distributed computing with high-speed optical networks. In *Proceedings of IEEE Computer*, volume 26, pages 8–18, 1993.
8. R. Dutta and N. Rouskas. Traffic grooming in WDM networks: Past and future. *IEEE Network*, 16(6):46–56, 2002.
9. M. Flammini, G. Monaco, L. Moscardelli, M. Shalom, and S. Zaks. Approximating the Traffic Grooming Problem in Tree and Star Networks. *Journal of Parallel and Distributed Computing*, 68(7):939–948, 2008.
10. M. Flammini, L. Moscardelli, M. Shalom, and S. Zaks. Approximating the Traffic Grooming Problem. *Journal of Discrete Algorithms*, 6(3):472–479, 2008.
11. L. Gargano and U. Vaccaro. *"Routing in All–Optical Networks: Algorithmic and Graph-Theoretic Problems" in: Numbers, Information and Complexity.* Kluwer Academic, 2000.
12. P. E. Green. *Fiber-Optic Communication Networks.* Prentice Hall, 1992.
13. S. Huang, R. Dutta, and G. 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.
14. R. Klasing. Methods and problems of wavelength-routing in all-optical networks. In *Proceedings of the MFCS Workshop on Communication*, pages 1–9, 1998.
15. Z. Li and I. Sau. Graph Partitioning and Traffic Grooming with Bounded Degree Request Graph. In *Proceedings of the 35th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 5911 of LNCS, pages 285–295, 2009.
16. X. Muñoz and I. Sau. Traffic Grooming in Unidirectional WDM Rings with Bounded Degree Request Graph. In *Proceedings of the 34th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 300–311, volume 5344 of LNCS, 2008.
17. R. Pulleyblank. *Faces of Matching Polyhedra.* PhD thesis, University of Waterloo, 1973.
18. R. Ramaswami. Multi-wavelength lightwave networks for computer communication. *IEEE Communications Magazine*, 31:78–88, 1993.
19. A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Springer, 2003.
20. K. Zhu and B. Mukherjee. A review of traffic grooming in wdm optical networks: Architecture and challenges. *Optical Networks Magazine*, 4(2):55–64, 2003.