Splitable Traffic Partition in WDM/SONET Rings to Minimize SONET ADMs

Gruia Călinescu * Peng-Jun Wan
Dept. of Computer Science
Illinois Institute of Technology
E-mail: {calinesc,wan}@cs.iit.edu

Abstract

SONET ADMs are the dominant cost factor in the WDM/SONET rings. Recently several articles [2, 3, 6, 10, 14] proposed a number of heuristics for traffic partition so as to use as few SONET ADMs as possible. Most of these heuristics assumes wavelength-continuity, i.e., the same wavelength is allocated on all of the links in the path established for a traffic stream. It was first observed and argued in [6] that the number of ADMs can be potentially reduced by allowing a traffic stream to be locally transferred from one ADM in a wavelength to another ADM in a different wavelength at any intermediate node, in other words, the traffic streams are splitable. In this paper, we study two variations of this minimum ADM problem with splitable traffic streams: all traffic streams have prespecified routings, and all traffic streams have no prespecified routings respectively. Both variations are shown to be NP-hard. For the former variation, a heuristic with approximation ratio at most 5/4 is proposed. For the latter variation, a similar heuristic with approximation ratio 3/2 is proposed.

Keywords: approximation algorithm, wavelength division multiplexing (WDM), optical networks, Synchronous Optical Networks (SONET), add-drop multiplexer (ADM).

*Work performed in part at Georgia Institute of Technology and supported in part by NSF grant CCR-9732746

1 Introduction

WDM/SONET (Wavelength Division Multiplexing/ Synchronous Optical NETworks) rings is a very attractive network architecture that is being deployed by a growing number of telecom carriers. In this network architecture, each wavelength channel carries a high-speed (e.g., OC-48) SONET ring [8]. The key terminating equipments are optical add-drop multiplexers (OADMs) and SONET add-drop multiplexers (ADMs). Each node is equipped with one OADM. The OADM can selectively drop wavelengths at a node. Thus if a wavelength does not carry any traffic from or to a particular node, the OADM allows that wavelength to optically bypass that node rather than being electronically terminated. Consequently, in each SONET ring a SONET ADM is required at a node if and only if it carries some traffic terminating at this node. Therefore, the SONET ADMs required by a set of traffic streams is determined by the routing and the wavelength assignment to them, i.e., a proper partition of the traffic streams into subsets such that each subset can be carried in a single wavelength. As the SONET ADMs are the dominant cost factor in the WDM/SONET rings, several articles [2, 3, 6, 10, 14] have proposed a number of heuristics for traffic partition so as to use as few SONET ADMs as possible. Most of these heuristics assumes wavelength-continuity, i.e., the same wavelength is allocated on all of the links in the path established for a traffic stream. Two versions were considered. In the first version, all traffic streams have predetermined routings and thus can be treated as circular arcs along a ring. This version can be described mathematically as follows:

- Instance: a set of circular-arcs A along a (clockwise) ring.
- Solution: a proper partition of A, $\Pi = \{A_1, A_2, \cdots, A_w\}$, such that for any $1 \leq i \leq w$ all arcs in each A_i are non-intersecting.
- Cost: the cost of each A_i is the number of different nodes of the ring that are the endpoints of the arcs in A_i, and the cost of the partition Π is the sum of the costs of A_i for all $1 \leq i \leq w$. The minimum cost over all proper solutions is called the minimum ADM cost of A.

In the second version, all traffic streams have no predetermined routings. Whether the underlying physical ring network is a bidirectional line-switched ring with two fibers (BLSR/2) or a bidirectional line-switched ring with four fibers (BLSR/4) [8], this version can be reduced to the following equivalent problem:

- Instance: a set of chords C along a (clockwise) ring.
- Solution: a proper partition of C, $\Pi = \{C_1, C_2, \cdots, C_w\}$, such that for any $1 \leq i \leq w$ all chords in each C_i can be routed as non-intersecting arcs over the ring.
- Cost: the cost of each C_i is the number of different nodes of the ring that are the endpoints of the chords in C_i, and the cost of the partition Π as the sum of the costs of C_i for all $1 \leq i \leq w$. The minimum cost over all proper solutions is called the minimum ADM cost of C.

Indeed, if each traffic stream is symmetrically duplex and its two portions in opposite directions must be routed along the same path (in opposite directions), we can treat the two working fiber rings as one (clockwise) ring, and each traffic stream as a (undirected) chord. Otherwise, each traffic stream has to be represented by a simplex traffic demand from its origin node to its destination node. Consider a simplex traffic demand from node i to node j. The above formulation uses instead a (undirected) chord with endpoints i and j. The ring of the above formulation is oriented clockwise, as all arcs are clockwise. Choosing the clockwise ring for the traffic demand corresponds to replacing the chord (i, j) with the arc with origin i and termination j, while choosing the counterclockwise ring for the demand corresponds to replacing the chord (i, j) with the arc with origin j and termination i. One can check that any valid solution in one formulation corresponds to a valid solution in the other formulation.

Both of the above two variations were proven to be NP-hard [3, 10]. The best known approximation ratios for both variations are both 3/2 [3].

Most previous theoretical on wavelength assignment in optical networks concentrated on minimizing the number of wavelengths, with or without routing [9, 12, 11], maximizing the number (or weighted profit) of traffic demands which can be routed using a given number of wavelengths [1], or minimizing the maximum load when routing is not prespecified [17], [13]. However, as argued in [6] and [4], often there are enough wavelengths, and in this case the goal should be minimizing the number of ADM devices.

Gerstel, Sasaki, and Ramaswami [7] showed that minimizing the number of SONET ADMs is intrinsically different from minimizing the number of wavelengths, and there exist cases where the two minima cannot be simultaneously achieved.

It was first observed and argued in [6] that the number of ADMs can be potentially reduced by allowing a traffic stream to be locally transferred from one ADM in a wavelength to another ADM in a different wavelength at any intermediate node; in other words, the traffic streams are splitable. For this splitable variation, we also consider two versions depending whether the traffic streams have prespecified routings or not. The arc version with splits can be stated as follows.

- Instance: a set of circular-arcs A along a (clockwise) ring.
- Solution: a choice of splitting each arc of A, thus obtaining A', and then a proper partition of A', $\Pi = \{A'_1, A'_2, \cdots, A'_w\}$, such that for any $1 \leq i \leq w$ all arcs in each A'_i are non-intersecting.
- Cost: the cost of each A'_i is the number of different nodes of the ring that are the endpoints of the arcs in A'_i, and the cost of the partition Π is the sum of the costs of A'_i for all $1 \leq i \leq w$. The minimum cost over all proper solutions is called the minimum ADM cost of A with splittings.

The chord version with splits can be stated as follows.

- Instance: a set of chords C along a (clockwise) ring.
- Solution: a choice of routing each chord of C, thus obtaining a set of arcs A; then a choice of splitting each arc of A, thus obtaining A'; and finally a proper partition of A', $\Pi = \{A'_1, A'_2, \cdots, A'_w\}$, such that for any $1 \leq i \leq w$ all arcs in each A'_i are non-intersecting.
- Cost: the cost of each A'_i is the number of different nodes of the ring that are the endpoints of the arcs in A'_i, and the cost of the partition Π is the sum of the costs of A'_i for all $1 \leq i \leq w$. The minimum cost over all proper solutions is called the minimum ADM cost of C with split.

Notice that in the chord version, we do not allow the transferring of a traffic from a ring in one direction to the other ring in the opposite direction for fault protection reasons. This restriction also make this formulation applicable to both BLSR/2 and BLSR/4, to both symmetric/full-duplex traffic and simplex traffic.

The benefit of splitting is illustrated by an example presented in [6] for the arc version with splits. In this example, the optimum with splits is 3/4 the optimum without splits. In fact, optimum with splits could be 2/3 the optimum without splits, as shown by the following example: $A = \{a_1, a_2, a_3\}$ over a 3-node ring, with $a_1 = (0, 2)$ (here (i, j) represents the arc from node i to node j), $a_3 = (2, 1)$, and $a_3 = (1, 0)$. The optimum without splits is 6, while with splits a solution of 4 can be obtained by splitting a_2 into $a_2' = (2, 0)$ and $a_2'' = (0, 1)$. After the splitting, an optimal partition consists of $\{a_1, a_2'\}$ and $\{a_2'', a_3\}$. See Figure 1.

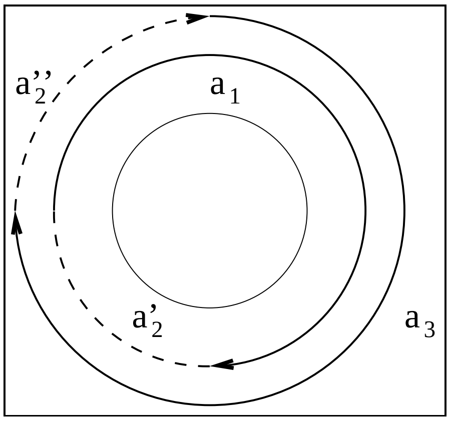

Figure 1: Arc a_2 is split into a_2' and a_2'' to obtain a solution with splits of cost 4.

Actually, the example above is extreme, in the sense that any solution with splits can be converted, by simply putting alone in a wavelength any arc which is split, into a solution without splits of cost at most 50% bigger. But how the two costs of optimum relate is not as interesting as how the best solutions we can find relate.

Based on the hardness result obtained in [3, 10], we show that finding an optimal solution with splits is also NP-hard, in both the prespecified and the non-prespecified routing versions. Thus we are interested in approximation algorithms. A heuristic called Cut-First was proposed in [6] for the arc version with splits. However, little is known about its approximation ratio. This paper introduces a new polynomial-time approximation algorithm for wavelength assignment to splitable lightpaths over WDM rings. The algorithm combines greedy ideas with Eulerian rounding, a technique similar to the Cut-First heuristic of [6]. We prove our algorithm has approximation ratio at most $5/4$. An example shows that the new algorithm has approximation ratio at least $10/9$. The Eulerian rounding technique can also be applied to the case when there is a choice for routing, yielding an algorithm with approximation ratio exactly $3/2$. By comparison, when lightpaths are not splitable, the best known algorithm [3] has approximation ratio in between $4/3$ and $3/2$.

This paper is organized as follows: in Section 2, we present the formal definition of the problem of assigning wavelengths to splitable lightpaths over WDM rings, with the objective of minimizing the number of SONET ADMs. In section 3 we present the proofs that both versions (routing is prespecified or not) of Ring Generation with Splitable Lightpaths are NP-hard. In Section 4 we present the new approximation algorithm and prove that its approximation ratio is in between $10/9$ and $5/4$. We also propose some practical improvements to the algorithm. In Section 5 we present the adaptation of our algorithm to the version when routing is not prespecified. We conclude with Section 6, in which we also discuss the simulation results of the new algorithm on some randomly generated instances.

We mention that a version of the new algorithm was presented, without proofs, in our survey paper [16].

2 Preliminaries

We assume that a (clockwise) ring network consists of n nodes numbered clockwise by $0, 1, \cdots, n-1$. All arithmetic involving nodes is performed implicitly using modulo n operations. The link from the node i to node $i+1$ in the ring is referred to as link i. A circular arc a over a ring is represented by an *ordered* pair $(o(a), t(a))$, where $o(a)$ is the origin of a and $t(a)$ is the termination of a. A chord in a ring is specified by an *unordered* pair (i, j) where i and j are the two endpoints of the chord.

Two arcs are said to be *intersecting* (or *overlapping*) if they contain a common link of the ring, *disjoint* if they do not contain any common node of the ring, *adjacent* if they are not intersecting but share at least one endpoint, and *complementary* if they are not intersecting and share two endpoints. A sequence of circular arcs is called a *chain* if the termination of each circular arc, except the last one, is the origin of the subsequent circular arc. If no two arcs in a chain overlap, the chain is called *valid*. If the termination of the last circular arc is the also the origin of the first circular arc, the chain is called *closed*, otherwise it is called *open*. The *length* of a chain P is the number of arcs in P and is denoted by $|P|$. A chain of length ℓ is called as an ℓ-chain. The *cost* of a chain is the number the nodes in the chain. For a closed chain, the length and cost coincide, while for an open chain, the cost is one more than the length.

Splitting a traffic stream corresponds to the following definition for splitting arcs: the arc a is replaced by several arcs $a_1, a_2, \cdots, a_k$ such that $o(a) = o(a_1)$, $t(a) = t(a_k)$, and $t(a_i) = o(a_{i+1})$ for any $1 \leq i < k$. To make notation simple, replacing an arc by itself is also a split, called a *nsplit*. An arc which is the result of an nsplit is called *original*. A split which is not a nsplit is called a *realsplit*. An arc which results from a realsplit is called *fragment*.

For both arc-version and chord version of the minimum ADM cost problem, with or without splits, we can restrict the solutions to partitions of a set of arcs into valid chains, referred to as *valid chain generations*. This restriction does not change the optimum value, but may require more wavelengths. The wavelength requirement by any solution can be further reduced by treating each chain as an arc and applying Tucker's algorithm for circular-arc coloring [15]. Notice that the cost of chains produced a valid chain generation is the number of input arcs or chords plus the number of open chains plus the number of splits.

More preliminary definitions must be introduced before presenting the algorithm. Given a set of arcs S, the *surplus* of a node i with respect to S is defined as follows:

$$sur_S(i) = |\{s \in S : t(s) = i\}| - |\{s \in S : o(s) = i\}|.$$

Note that the surplus might be negative and in fact $\sum_{i=0}^{n-1} sur_S(i) = 0$. An open chain $P = \langle a_1, a_2, \cdots, a_k \rangle$ in S such that $sur_S(o(a_1)) < 0$ and $sur_S(t(a_k) > 0$ is called *tight with respect to S*, or simply *tight*, when S is understood as being a current set of arcs. The *deficiency* of a node i is $def_S(i) = \frac{1}{2}|sur_S(i)|$. The deficiency of a set of arcs S is $def(S) =$

$\sum_{i=0}^{n-1} def_S(i)$. Then $|S| + def(S)$ is a lower bound on the minimum ADM cost required by S.

Given a set of chords C, the *deficiency* of a node i is 0 if the node has even degree, 1/2 if the node has odd degree. The deficiency of a set of chords C is defined as follows: $def(C) = \sum_{i=0}^{n-1} def_C(i)$. Then $|C| + def(C)$ is a lower bound on the minimum ADM cost required by S.

For the purpose of the proof, we use arcs which are not in A. Such an auxiliary arc is called a *fake*, and has an origin and a termination like any other arc. We divide the set of arcs in two: *red* arcs and *blue* arcs. An arc is red if it does contain the link $n-1$, and blue otherwise. Given a set of arcs S, the *blue number* of S, denoted by $b(S)$, is the number of blue arcs in S. Sometimes we write $b(a)$ instead of $b(\{a\})$. Note that any valid closed chain has blue number one, while a valid open chain has blue number at most one.

3 NP-Hardness Results

The NP-hardness proofs in [3, 10] implies that it is NP-Complete to determine, for both the arc-version and chord version of the minimum ADM cost problem with splits, whether the optimum equals the number of arcs or chords respectively. Notice that for the arc-version with splits, optimum equals the number of arcs if and only if the set of arcs can be partitioned into valid closed chains without any splits; and for the chord-version with splits, again optimum equals the number of chords if and only if the set of chords can be oriented and then partitioned into valid closed chains without any splits. Thus we immediately obtain the following hardness result:

Theorem 1 *Both the arc-version and chord version of the minimum ADM cost problem with splits are NP-hard.*

4 The Arc-Version with Splits

We start by describing the last and most interesting phase of the algorithm, *Eulerian rounding*. Let S be a set of arcs. We consider two cases. In the first case, $def\,(S) > 0$. We first add a set of $def\,(S)$ fake arcs F such that $def(S \cup F) = 0$. This can be easily done by adding one by one fake arcs with the origin being a node of positive surplus and the termination being a node of negative surplus, thus each fake arc decreasing the deficiency by one. Now the directed graph with edges $S \cup F$ is Eulerian. Choosing any Eulerian tour and then removing all fake arcs results in $def(S)$ open chains. For every invalid (open) chain P, break it into valid chains as follows (see Figure 2): for each circular arc a in P that passes through $o\,(P)$, the origin of P, split it into two arcs

$$a' = (o\,(a)\,, o\,(P))\,, a'' = (o\,(P)\,, t\,(a))\,.$$

After these splittings, the invalid chain P is then be decomposed into valid chains by walking along P from $o\,(P)$ and output a valid chain whenever reaching $o\,(P)$.

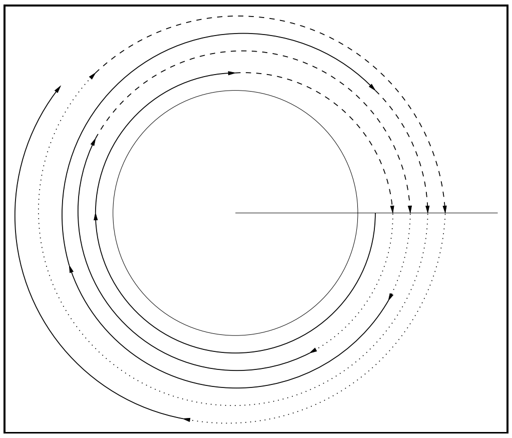

Figure 2: Eulerian rounding. Arcs are split into two fragments when they pass through node $t(a_1)$: the dashed fragment closes a chain, while the dotted fragment starts another chain.

In the second case, $def\,(S) = 0$, and thus the directed graph with edges S is Eulerian. Choosing any Eulerian tour. Let i be any node which is the origin of some arc. For any circular arc a in the oriented Eulerian tour that passes through i, split it into two arcs

$$a' = (o(a), i), a'' = (i, t(a)).$$

After these splittings, the oriented Eulerian tour is then be decomposed into valid (closed) chains by walking along the oriented Eulerian tour from node i and output a valid (closed) chain whenever reaching node i.

Lemma 2 *The solution produced from Eulerian rounding for S has cost at most $|S| + b(S) + def(S)$.*

Proof. We first consider the case that $def(S) > 0$. Let P be any invalid chain. Note that among all valid chains generated by Eulerian rounding from P, exactly one is open and all others are closed. Since the chain P must pass trough the link $(n-1, 0)$ each time before passing trough $o(P)$, the number of splits is at most $b(P)$. So the total cost of these valid chains generated from P is at most $|P| + b(P) + 1$. Since there are total $def(S)$ open chains, the total cost of all valid chains is at most $|S| + b(S) + def(S)$.

Now we consider the case that $def(S) = 0$. In this case, all valid chains are closed, and the number of splits is at most $b(P)$. So the total cost of these valid chains is at most $|S| + b(S)$. ■

Using the machinery developed before, we proposed the following approximation algorithm for Wavelength Assignment in WDM Rings with Splitable Lightpaths:

- The input is a set of arcs A.
- Phase 1: While A contains a valid closed chain P of length two, nsplit the arcs in P, output the valid (closed) chain consisting of the two arcs, and set $A \leftarrow A \setminus P$.
- Phase 2: While A contains a valid closed chain P of length three, nsplit the arcs in P, output the valid (closed) chain consisting of the three arcs, and set $A \leftarrow A \setminus P$.
- Phase 3: While such an arc exists, select $a \in A$ such that a is blue and such the chain $\langle a \rangle$ is tight. Nsplit a, output the valid (open) chain consisting of a, and set $A \leftarrow A \setminus \{a\}$.

- Phase 4: While such a pair of arcs exists, select $a_1, a_2 \in A$ such that $\langle a_1, a_2 \rangle$ is a tight valid chain, and one of a_1, a_2 is blue. Nsplit a_1 and a_2, output the valid (open) chain consisting of the two arcs, and set $A \leftarrow A \setminus \{a_1, a_2\}$.
- Phase 5: Do the Eulerian rounding of A.

The last phase is the most interesting one. If optimum is 'perfect', (has only valid closed chains and no splits), Eulerian rounding cannot guarantee a good ratio. This is the reason why we use the first two phases, which achieve good results when there are many short valid closed chains. The third and fourth phase produce open chains, but decrease the deficiency. Since in any solution, the number of open chains is at least the deficiency, it makes sense to reduce deficiency. We also insist on reducing the number of blue arcs while reducing deficiency in phases three and four, to obtain a good Eulerian rounding in the fifth phase according to Lemma 2.

Before the proof that the algorithm has approximation guarantee 5/4, we introduce some notation. For a set of arcs A, we use $opt(A)$ to denote the value of the optimum solution to Wavelength Assignment in WDM Rings with Splitable Lightpaths which has A as the input. If A is understood, we use only *opt*.

The first phase is very intuitive. Indeed, let B be the set of arcs nsplit and then assigned to valid chains during the first phase. Then there is always an optimum solution which assigns to valid chains the arcs of B exactly the way our algorithm does. This fact follows from the following lemma, which was previously known, but whose proof we include for completeness.

Lemma 3 *Let A be a set of arcs and a_1, a_2 be two arcs of A which form a valid closed chain. Then $opt(A) = opt(A \setminus \{a_1, a_2\}) + 2$.*

Proof. Assume we have a solution which does not put a_1 and a_2 in the same valid closed chain. Let P_1 be the chain containing a_1 and P_2 be the chain containing a_2. Removing a_1 from P_1 leaves us with one of the following two cases:

1. If P_1 is closed: one valid chain, which we call P_1^*, from $t(a_1)$ to $o(a_1)$.

2. If P_1 is open: two non-overlapping valid chains, which we call P_1' and P_1'', and which both could be empty. P_1' starts at $t(a_1) = o(a_2)$ and ends at some node in the interior of a_2, and P_1'' starts at some node in the interior of a_2 and ends at $o(a_1) = t(a_2)$.

Similarly, removing a_2 from P_2 leaves us with either P_2^* or two chains P_2' and P_2''. Then, after putting a_1 and a_2 in the same closed valid chains, depending on which case we are in, we produce one of the following four combinations:

1. P_1^* and P_2^* in one valid closed chain, at a cost of $|P_1^*| + |P_2^*| = |P_1| + |P_2| - 2$. The total cost of P_1 and P_2 is $|P_1| + |P_2|$.

2. P_2'', P_1^*, P_2' in valid open chain, at a cost of $|P_2''| + |P_1^*| + |P_2'| + 1 = |P_1| - 1 + |P_2|$. The total cost of P_1 and P_2 is $|P_1| + |P_2| + 1$.

3. P_1'', P_2^*, P_1' in one valid open chain, at a cost of $|P_1''| + |P_2^*| + |P_1'| + 1 = |P_1| - 1 + |P_2|$. The total cost of P_1 and P_2 is $|P_1| + |P_2| + 1$.

4. P_1'' and P_2' in one valid open chain, and P_2'' and P_1' in another valid open chain, at a total cost of at most $|P_1''| + |P_2'| + 1 + |P_2''| + |P_1'| + 1 = |P_1| + |P_2|$. The total cost of P_1 and P_2 is $|P_1| + |P_2| + 2$.

In all the cases, the cost of the solution which puts a_1 and a_2 in the same valid closed chain is at most the cost of the solution which does not put a_1 and a_2 in the same valid closed chain. ■

Based on the lemma above, in the following we assume that no two arcs of the input form a valid closed chain.

Theorem 4 *The algorithm above has performance ratio at most* $5/4$.

Proof.

First we present a general overview of the proof. We fix OPT, an optimum solution, and based on OPT we give credit to arcs of A. Recall that A is the input. Also, each node i gets $def_A(i)$ credit. The exact initial credit allocation scheme is described later.

Before the algorithm starts, as shown in Lemma 5, the total credit is at most $(5/4)opt$. During the execution of phases two, three and four of the algorithm, as the algorithm selects arcs, puts them in valid chains and outputs the chains, (during these phases we only nsplit), we take the credit from these arcs, and in phases three and four, also from the nodes whose deficiency decreases. We use this credit taken to cover the cost of the valid chains the algorithm produces and to give some extra credit to some other arcs, according to a credit transfer scheme to be described later.

Finally, by the time we get to the fifth phase, the total credit remaining covers the cost of the output of Eulerian rounding.

We start the proof somehow backwards, discussing this last phase. This would give the motivation for all the previous steps. Let S be the set of remaining arcs after phase four. Assume each node i has $def_S(i)$ and each arc $a \in S$ has $1 + b(a)$ credit. In total, we have $def(S) + |S| + b(S)$ credit and therefore, by Lemma 2, we can cover the cost of the Eulerian rounding. All the remaining discussion is about how to make sure that at the end of phase four, each remaining arc a has $1 + b(a)$ credit and each node i has $def_S(i)$ credit.

We now describe the initial credit allocation. For a set of arcs S, we define OPT_S to be the restriction of OPT to the set of arcs S. Each node i gets $def_A(i)$ credit. Every red arc receives 1 credit. Blue arcs receive credit according to their position in OPT, as given by the following scheme.

- Initialize S to be the initial set of arcs, A.
- Phase 1: As long as possible, do the following: if in OPT_S there is a (valid) tight (with respect to S), open chain $P = \langle a \rangle$, where a is blue and nsplit, then a receives 1.5 credit. Remove a from S.
- Phase 2: As long as possible, do the following: if in OPT_S there is a (valid) tight (with respect to S), open chain $P = \langle a_1, a_2 \rangle$, where either a_1 or a_2 is blue, and both are nsplit, the blue arc receives 1.75 credit. Remove from S the arcs of P.
- Phase 3: As long as possible, do the following: if in OPT_S there is a (valid) closed chain $P = \langle a_1, a_2, a_3 \rangle$, where a_1, a_2 and a_3 are nsplit, the blue arc in P receives 1.75 credit. Remove from S the arcs of P.
- Every remaining blue arc receives 2 credit.

Lemma 5 *The total initial credit is at most $\frac{5}{4}opt$.*

Proof. We consider the chains of OPT one by one. We start with the chains used in the first three phases of the credit allocation scheme above, in the order they are used. Then we continue with the remaining chains of optimum, in some order to be described later.

Take an open chain of OPT as described in the first phase of the scheme. For the arc a, OPT incurs a cost of 2. We give 1.5 credit to a, 0.5 credit to $o(a)$, and 0.5 credit to $t(a)$. Altogether, we give 5/4 times the cost of this chain in OPT. Note that after removing each such arc a from S, both $def_S(t(a))$ and $def_S(o(a))$ drop by 0.5.

Take an open chain of OPT as described in the second phase of the scheme. For the chain $P = \langle a_1, a_2 \rangle$, OPT incurs a cost of 3. We give 1 credit to the red arc, 1.75 credit to the blue arc, 0.5 credit to $t(a_2)$ and 0.5 credit to $o(a_1)$. Altogether, we give 3.75 credit, which is 5/4 times the cost of this chain in OPT. Note that after removing the arcs from P from S, both $def_S(t(a_2))$ and $def_S(o(a_1))$ drop by 0.5.

Take a closed chain of OPT as described in the third phase of the scheme. For the chain $P = \langle a_1, a_2, a_3 \rangle$, OPT incurs a cost of 3. We give 3.75 credit, which is 5/4 times the cost of this chain in OPT. Note that no deficiency changes after removing the arcs from P from S.

Now we start giving credit to the arcs in the set S of remaining arcs, and we also give more credit to the nodes. As described in the scheme, each blue arcs receives 2 credit and each red arcs receives 1 credit. In addition we need to give to each node i exactly $def_S(i)$ credit, as i already received $def_A(i) - def_S(i)$ credit.

Take a closed chain P of OPT_S, consisting only of nsplit arcs. Then the cost of P in OPT is $|P|$. P contains exactly one blue arc, and therefore we give $|P|+1$ credit to the arcs of P. As P was not processed in phase 3 of the scheme, and since we assumed P cannot have only two arcs, P has at least four arcs. Therefore we have $|P| + 1 \leq \frac{5}{4}|P|$. Remove the arcs of P from S and note that no deficiency changes as a result of this removal. Repeat the step above as long as possible.

Consider an arc $a \in A$ which is realsplit into $d_1, d_2, \cdots, d_k$. Then a will get enough (at least $1 + b(a)$) credit if each d_i gets at least $0.5 + b(d_i)$ credit. Let S' be the set obtained from S by the splitting optimum does. Note that for any node i, $def_{S'}(i) = def_S(i)$. We continue a credit-giving scheme, using the arcs of S'. It is enough if we give at least $def_{S'}(i)$ credit to each node i, at least $1 + b(a)$ to each original arc a, and at least $0.5 + b(d)$ to each fragment arc d. That is exactly what we are going to do.

Take a closed chain P of $OPT_{S'}$. By this stage, P has some fragment arc. The cost in OPT of P is $|P|$, while we give credit at most $|P| + 0.5$, as exactly one arc in P is blue, but at least one of the arcs of P, being a fragment, saves 0.5 credit. Since $|P| \geq 2$, we have $|P| + 0.5 \leq \frac{5}{4}|P|$, and therefore the credit we give is at most 5/4 times the cost of this chain in OPT. Remove the arcs of P from S' and note that no deficiency changes as a result of this removal. Repeat the step above as long as possible.

At this moment, $OPT_{S'}$ has only open chains.

Take a tight (with respect to S') open chain $P = \langle a_1, a_2, \cdots, a_k \rangle$ of $OPT_{S'}$, consisting only of original (nsplit) arcs. The cost in OPT of P is $|P| + 1$. If P has no blue arc, we give $|P|$ credits to the arcs of P and 0.5 credit to $o(a_1)$ and 0.5 credit to $t(a_k)$, for a total of $|P| + 1$ credit. In the following we assume P has blue arcs. Since P is valid, it has exactly one blue arc. Since P was not considered in phases one and two of the scheme, $|P| \geq 3$. Therefore, we give $|P| + 1$ credit to the arcs of P. We also give 0.5 credit to $o(a_1)$ and 0.5 credit to $t(a_k)$. In total, we give $|P| + 2$ credit, which is at most $\frac{5}{4}(|P| + 1)$. Remove the arcs of P from S' and note that after the removal, both $def_{S'}(o(a_1))$ and $def_{S'}(t(a_k))$ drop by 0.5. Repeat the step above as long as possible.

Take a tight (with respect to S') open chain $P = \langle a_1, a_2, \cdots, a_k \rangle$ of $OPT_{S'}$, where at least one of the $a_i's$ is a fragment arc. The cost in OPT of P is $|P| + 1$. We give credit at most $|P| + 0.5$ to the arcs of P, 0.5 to $o(a_1)$, and 0.5 to $t(a_k)$. In total, we give $|P| + 1.5$ credit, which is at most $\frac{5}{4}(|P| + 1)$, as $|P| \geq 1$. Remove the arcs of P from S' and note that after the removal, both $def_{S'}(o(a_1))$ and $def_{S'}(t(a_k))$ drop by 0.5. Repeat the step above as long as possible.

Take a sequence of open chains $P_1, P_2, \cdots, P_k$ of $OPT_{S'}$, which concatenated form a (non-valid) closed chain. The cost in OPT of P_i is $|P_i| + 1$, and the total cost of the chains $P_1, P_2, \cdots, P_k$, is $k + \sum_{i=1}^{k} |P_i|$. We give $1 + b(a)$ to any arc (original or fragment) in any of these k open chains, at a total cost of at most $k + \sum_{i=1}^{k} |P_i|$ (we used the fact that each P_i has at most one blue arc). Remove the arcs of $P_1, P_2, \cdots, P_k$ from S' and note that after the removal, no deficiency changes. Repeat the step above as long as possible.

Take a sequence of open chains $P_1, P_2, \cdots, P_j$ of $OPT_{S'}$, which concatenated form a (non-valid) tight open chain $P' = \langle a_1, a_2, \cdots, a_k \rangle$. Note that $j > 1$, since if such a chain P_1 existed, would have been considered previously. This implies $|P'| > 1$ Also note that $b(P') \leq j$. The cost in OPT of P' is $|P'| + j$. We give credit $1 + b(a_i)$ to each a_i, 0.5 to $o(a_1)$, and 0.5 to $t(a_k)$. In total, we give at most $|P'| + j + 1$ credit, which is at most $\frac{5}{4}(|P'| + j)$, since $|P'| + j \geq 4$. Remove the arcs of P' from S' and note that after the removal, both $def_{S'}(o(a_1))$ and $def_{S'}(t(a_k))$ drop by 0.5. Repeat the step above as long as possible.

It is easy to see that $S' = \emptyset$ at this moment. This completes the proof of Lemma 5. ■

We continue with the proof of Theorem 4. But first some definitions. If a blue arc has 2 credit, it is *happy*, otherwise it is *unhappy*. As our credit transfer scheme only takes credit from an arc when the arc is put (in phases 2, 3, and 4 of the algorithm) in a valid chain (which is then output), happy arcs stay happy as long as they are in the current set of arcs. By the initial credit allocation scheme, we have three types of unhappy arcs:

1. Blue arcs given 1.5 credit during the first phase of the initial credit allocation scheme. These arcs are called *unhappy singles*.
2. Blue arcs given 1.75 credit in the second phase of the initial credit allocation scheme. The two arcs from the same open chain of OPT are called *partners*, and they form an *unhappy pair*.
3. Blue arcs given 1.75 credit in the third phase of the initial credit allocation scheme. The three arcs from the same closed chain of OPT are called *partners*, and they form an *unhappy triple*.

For a node i, define $dep(i)$ to be the number of unhappy singles a with $o(a) = i$ plus the number of unhappy pairs $\langle a_1, a_2 \rangle$ with $o(a_1) = i$. As arcs are made happy or assigned to valid chains (which are then output), $dep(i)$ is decreasing during the execution of the algorithm. Similarly, for a node i, define $arr(i)$ to be the number of unhappy singles a with $t(a) = i$ plus the number of unhappy pairs $\langle a_1, a_2 \rangle$ with $t(a_2) = i$. As arcs are made happy or assigned to valid chains (which are then output), $arr(i)$ is decreasing during the execution of the algorithm.

Since only tight (with the respect to the current set of arcs during the credit allocation scheme) chains contribute to arr, initially $arr(i) \leq sur(i)$. Similarly, only nodes with initial negative surplus can have nonzero dep, and in fact initially $dep(i) \leq -sur(i)$.

Our credit transfer scheme maintains the following *deparr* invariant during the execution of the algorithm (that is, with respect to the current set of arcs): for any node i, if $arr(i) > 0$, then $arr(i) \leq sur(i)$, and if $dep(i) > 0$, then $dep(i) \leq -sur(i)$.

To maintain the deparr invariant, our credit transfer scheme ensures that whenever (during phases three or four) we choose an open chain, and we increase the surplus of a node i with $dep(i) > 0$, we also decrease $dep(i)$ by either making a unhappy single happy or by making the blue arc of an unhappy pair happy. Similarly, whenever (during phases three or four) we choose an open chain, and we decrease the surplus of a node i with $arr(i) > 0$, we also decrease $arr(i)$ by either making a unhappy single happy or by making the blue arc of an unhappy pair happy.

Our credit transfer scheme, as we will see, also makes sure that whenever an arc from an unhappy pair or from an unhappy triple is selected and used by the algorithm, any remaining blue arc from the pair or triple receives enough credit to become happy.

Phase two of the algorithm is designed to eliminate the unhappy triples. Please note that no surplus is changed, and therefore there is no need to worry about arr or dep.

Our credit transfer scheme is as follows: Let $P = \langle a_1, a_2, a_3 \rangle$ be a closed chain selected in phase two of the algorithm, and assume that a_1 is the blue arc. The cost of P in the output of our algorithm is 3. The blue arc a_1 has at least 1.5 credit, and therefore the total credit on these three arcs is at least 3.5. We use 3 credit to cover the output, give to the unhappy partner (if it exists) of a_2 0.25 credit, and give to the unhappy partner (if it exists) of a_3 0.25 credit, thus making happy any remaining unhappy arc which had one of its partners selected. No unhappy triple remains at the end of the second phase.

We proceed to the third phase of the algorithm. A blue arc a is selected such that $sur(o(a)) < 0$, $sur(t(a)) > 0$. The algorithm puts a into a valid chain by itself, and outputs this chain of cost 2. We get whatever credit a has, and another 1 credit since we decrease $def(o(a))$ and $def(t(a))$. We need to pay for the output, for unhappy partners separated, and for maintaining the deparr invariant.

We have three cases: a is an unhappy single, a is a happy blue arc, or a is a partner in an unhappy pair. If a is an unhappy single, as both $dep(o(a))$ and $arr(t(a))$ are decreasing by 1, the deparr invariant is maintained, and no one is separated, and so we are done.

If a is a happy arc, it has 2 credit. After covering the output, 1 credit is left. No partners are separated. However we might need to pay to maintain the deparr invariant, which could be become violated at both $o(a)$ and $t(a)$. We allocate 0.5 to fixing each $o(a)$ and $t(a)$. 0.5 credit is enough to make any unhappy arc happy, and therefore it is enough to make an unhappy single or pair happy, thus decreasing $dep(o(a))$ or $arr(t(a))$.

Now assume a is in an unhappy pair. By symmetry, say $\langle a, d\rangle$ is the unhappy pair (the other case being $\langle d, a\rangle$ as the unhappy pair). After covering the output, 0.75 credit is left. Now since the unhappy pair $\langle a, d\rangle$ disappears, $dep(o(a))$ decreases and therefore the deparr invariant is maintained at $o(a)$. We allocate 0.5 credit to fixing the deparr invariant at $t(a)$, and this credit is enough by the same argument used in the case when a is happy.

Finally, no unhappy single arc a can survive the third phase of the algorithm, as shown below. Since $dep(o(a))$ is positive (because of a itself), by the deparr invariant, $sur(o(a)) < 0$, and similarly $sur(t(a)) > 0$. Therefore a is eligible to be selected by the algorithm. Thus at the end of the third phase, only unhappy pairs could exist.

We now analyze phase four. Due to the deparr invariant, each unhappy pair is eligible to be chosen by the algorithm in the fourth phase, as long as it is unhappy. A valid open chain $\langle a_1, a_2\rangle$ is selected such that $sur(o(a_1)) < 0$ and $sur(t(a_2)) > 0$, and either a_1 or a_2 is blue. This open chain is output, and has a cost of 3. We get whatever credit a_1 and a_2 have, and another 1 credit since we decrease $def(o(a))$ and $def(t(a))$. We need to pay for the output, for unhappy partners separated, and for maintaining the deparr invariant.

After paying for the output, we have at least 0.75 credit in hand. We allocate 0.25 credit to fix the deparr invariant at each of $o(a_1)$ and $t(a_2)$, and 0.25 to make happy the blue partner (if any) of the red arc in $\{a_1.a_2\}$. As at this stage, any unhappy blue arc already has 1.75 credit, 0.25 credit is enough to make it happy.

Finally, note that as long as an unhappy pair exists, the algorithm is still in phase four as an eligible pair exists. This fact implies that at the end of phase four, no unhappy arc exists. Thus we have enough credit for the fifth and last phase. This completes the proof of Theorem 4. ■

Obvious practical improvements are: after Phase 4, but before the Eulerian rounding, insert three other phases:

- Phase 4.1. While A contains a valid closed chain P, find one with minimum number of arcs. Nsplit the arcs in P, output P, and set $A \leftarrow A \setminus P$.
- Phase 4.2 While A contains a tight valid chain $P = \langle a_1, \cdots, a_k \rangle$, nsplit the arcs in P, output P, and set $A \leftarrow A \setminus P$.
- Phase 4.3 While A contains a closed chain P, find one with $b(P)$ minimum. The Floyd-Warshall algorithm on the graph with vertex set N and arc set A finds this minimum. Do an Eulerian rounding on P (producing $b(P)$ closed chains). Then set $A \leftarrow A \setminus P$.

And after Eulerian rounding, the three transformations (Merging, Combining and Splitting) proposed by [6] should also be applied, if possible.

There is nothing special about the link $n-1$ deciding which arc is blue and which one is not. If enough computing time is available, one should try running the algorithm using each of the remaining $n-1$ links of the ring to decide which arc is blue and which one is not.

We are not able to prove a better approximation ratio based on these improvements. The worst example (for which the practical improvements also do not help) we have is the following: $n = 6$ and A consists of the following nine arcs:

$$\begin{aligned} a_1 = (0,2), a_2 = (2,5), a_3 = (5,0), \\ b_1 = (2,4), b_2 = (4,1), b_3 = (1,2), \\ c_1 = (4,0), c_2 = (0,3), c_3 = (3,4). \end{aligned}$$

The optimum produces the closed chains

$$\langle a_1, a_2, a_3 \rangle, \langle b_1, b_2, b_3 \rangle, \langle c_1, c_2, c_3 \rangle,$$

of total cost 9. Indeed, the optimum has cost 9, while if our algorithm is unlucky and chooses the closed chain $\langle a_1, b_1, c_1 \rangle$ in phase two, it produces a solution of cost 10. See Figure 3.

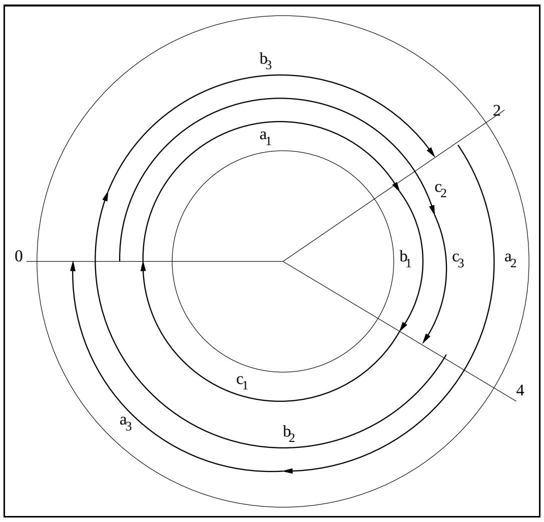

Figure 3: The algorithm selects the closed chain $\langle a_1, b_1, c_1 \rangle$. Eulerian rounding then splits one arc in two (a_2, for example) and produces a solution of cost 10. Optimum uses the closed chains $\langle a_1, a_2, a_3 \rangle$, $\langle b_1, b_2, b_3 \rangle$, and $\langle c_1, c_2, c_3 \rangle$, of total cost 9.

5 The Chord-Version with Splits

In this section we describe the adaptation of our algorithm to the version when routing is not prespecified. The input is given as a set of chords. Obviously, the optimum with splits is at most the optimum without splits. Actually, the optimum with splits can be 25% lower, as shown by the following example: let $n = 5$ and $C = \{c_i : 1 \leq i \leq 5\}$ with $c_1 = (0,3), c_2 = (3,1), c_3 = (1,4), c_4 = (4,2), c_5 = (2,0)$. The optimum without splits is 8, while with splits a solution of cost 6 can be obtained by splitting c_3 into two chords

$$c_3' = (1,0), c_3'' = (0,4),$$

and then c_1, c_2, c_3' form one valid chain and c_3'', c_4, c_5 form another valid chain. See Figure 4. Note that the example giving a 33% potential saving for splitting arcs (prespecified routing) does not hold for chords.

Our algorithm is based on Eulerian rounding, as follows:

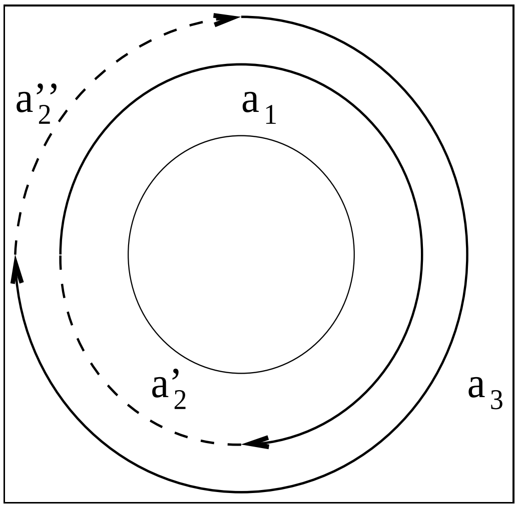

Figure 4: Chord c_3 is split into c_3' and c_3'' to obtain a solution with splits of cost 6.

- The input is a set of chords C.
- Add a set of fake chords H such that $|H| = def(C)$ and $def(C \cup H) = 0$. This can be easily done by adding one by one fake chords in between nodes of odd degree.
- Find an Eulerian cycle EC of $C \cup H$. EC can be traversed in two directions, and each direction transforms the chords of $C \cup H$ into arcs. We call A' and A'' the sets of arcs obtained from C this way. If $b(A') > b(A'')$, let A be A'', otherwise let $A = A'$. C is oriented to obtain A.
- Do the Eulerian rounding of A.

For a chord $c \in C$, we call $a_c' \in A'$ and $a_c'' \in A''$ the corresponding arcs obtained by orienting c in opposite directions. Exactly one of a_c' and $a_c'' \in A''$ is blue, and therefore $b(A') + b(A'') = |C|$. We conclude that $b(A) \leq |C|/2$. Note that $def(A) = |H| = def(C)$. By lemma 2, with Eulerian rounding of A we obtain a solution of cost at most $|A| + b(A) + def(A) \leq \frac{3}{2}|C| + def(C)$. We have:

Theorem 6 *The above algorithm has approximation ratio exactly* $3/2$.

Proof. The fact that the approximation ration is at most $3/2$ follows from the discussion above. The following example shows that the approximation ratio of the algorithm is at least $\frac{3}{2}$.

Let n, the number of nodes on the ring, be odd. Consider the following set of chords: $C = \{c_0, c_1, \cdots, c_{n-3}, c', c''\}$, where for $0 \le i \le n-3$, the endpoints of c_i are the nodes i and $i+2$, the endpoints of c' are 0 and 1, and the endpoints of c'' are $n-2$ and $n-1$.

The Eulerian tour selected by the algorithm is

$$c_{n-3}, c_{n-5}, \cdots, c_0, c', c_1, c_3, \cdots, c_{n-4}, c''.$$

The chords $c', c_1, c_3, \cdots, c_{n-4}, c''$ are oriented clockwise, and the chords $c_{n-3}, c_{n-5}, \cdots, c_0$ are oriented counterclockwise. For Eulerian rounding, node 0 is selected. Then all the arcs $c_{n-3}, c_{n-5}, \cdots, c_2$ are going to be split in step 7, obtaining a solution of cost $n + \frac{n-3}{2}$, since there are n chords and $\frac{n-3}{2}$ of them are split.

However, an optimum of cost $n+2$ exists: orient all the chords clockwise and produce two open chains: $c_0, c_2, \cdots, c_{n-3}$ and $c', c_1, c_3, \cdots, c_{n-4}, c''$. Therefore, the cost ratio of the output of the algorithm to the optimum is $\frac{n+(n-3)/2}{n+2}$. If we let n large enough, this ratio gets arbitrarily close to $\frac{3}{2}$.

In conclusion, the approximation ratio of the algorithm presented in this section is exactly $\frac{3}{2}$. ■

An obvious practical improvement is: once an orientation has been selected, run the algorithm from Section 4 on the given set of arcs. Another practical improvement would be the equivalent of Phases 4.1 and 4.3 of the algorithm from Section 4 applied directly to chords, before Eulerian rounding. That is, before an orientation is selected, find cycles in C such that, when oriented in the best of the two possible ways, they have small blue number (recall that a closed chain with blue number one is valid). Then do an Eulerian rounding on the cycle, and remove the cycle from C. A procedure to find in polynomial time a cycle which can be oriented with blue number one, if such a cycle exists, is described in [3]. This procedure can be adapted to find in time polynomial in n^k a cycle which can be oriented with blue number k, or report that no such cycle exists. We do not know, however, how to find the smallest blue number possible. None of the practical improvements above helps in the example from the proof of Theorem 6.

6 Conclusions

We proved that the approximation ratio of the algorithm, in the prespecified routing version, is in between 10/9 and 5/4. It will be interesting to find the exact worst-case behavior of the algorithm. We do not have a proof that minimum ADM cost problem (in any of the four versions: arc/chord, splitable/non-splitable) is MAX-SNP Hard and therefore the existence of a polynomial-time approximation scheme remains open.

We implemented the algorithm in C++ and run it on several randomly generated instances, with at most 160 nodes and 7000 arcs. The code is available upon request. Not being able to compute the optimum, we used as lower bound the number of arcs plus the deficiency. The output of the algorithm was between 7% and 15% more than the lower bound. We also implemented the more natural variation which uses Phases 4.1 and 4.3 before Phase 3. This natural variation, for which we could not prove a good approximation ratio, gave slightly better results on about half the instances.

References

[1] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A Unified Approach to Approximating Resource Allocation and Scheduling In *Proc. STOC 2000*

[2] G. Belvaux, N. Boissin, A. Sutter, and L.A. Wolsey, "Optimal placement of add/drop multiplexers: static and dynamic models", *European Journal of Operations Research*, vol. 108, iss. 1, pg. 26-35, 1998.

[3] G. Călinescu and P.-J. Wan, "Traffic Partition in WDM/SONET Rings to Minimize SONET ADMs", submitted for publication.

[4] A.L. Chiu and E.H. Modiano. Traffic Grooming Algorithms for Reducing Electronic Multiplexing Costs in WDM Ring Networks. In *Journal of Lightwave Technology*, Vol. 18, pages 2-12.

[5] M. Garey, D. Johnson, G. Miller, and C. Papadimitriou, The complexity of coloring circular arc graphs and chords, *SIAM Journal of Discrete Math*, vol. 1, no. 2, pg. 216-227, 1980.

[6] O. Gerstel, P. Lin, and G. Sasaki, "Wavelength assignment in a WDM ring to minimize cost of embedded SONET rings", Proc. IEEE INFOCOM'98, Volume: 1 , Page(s): 94 -101 vol.1

[7] O. Gerstel, G. Sasaki, and R. Ramaswami, "Cost effective traffic grooming in WDM rings", Proc. IEEE INFOCOM'98. Volume: 1 , Page(s): 69 -77.

[8] I. Haque, W. Kremer, and K. Raychauduri, "Self-Healing Rings in a synchronous environment", SONET/SDH: a sourcebook of synchronous networking, Eds. C.A. Siller and M. Shafi, IEEE Press, New York, pg. 131-139, 1996.

[9] V. Kumar. Approximating Circular Arc Coloring and Bandwidth Allocation in All-Optical Ring Networks. In *Proc. APPROX '98*, pages 147-158.

[10] L.W. Liu , X.-Y. Li, P.-J. Wan, and O. Frieder, "Wavelength Assignment in WDM Rings to Minimize SONET ADMs", *Proc. INFOCOM 2000*, Vol. 2, pg. 1020 –1025.

[11] M. Mihail, C. Kaklamanis, and S. Rao. Efficient Access to Optical Bandwidth. In *Proc. FOCS '95*, pages 548-557.

[12] P. Raghavan and E. Upfal, Efficient routing in all-optical networks, in Proc. 26 th ACM Symp. Theory of Computing, pg. 134-143, 1994.

[13] A. Schrijver, P.D. Seymour, and P. Winkler, " The Ring Loading Problem", *SIAM J. Disc. Math.*, Vol. 11, No. 1, pg. 1-14, February 1998.

[14] A. Sutter, F. Vanderbeck, and L. Wolsey, "Optimal placement of add/drop multiplexers: heuristics and exact algorithms", *Operations Research*, vol. 46, iss. 5, pg. 719-728, 1998

[15] A. Tucker, "Coloring a family of circular arcs", SIAM Journal of Applied Math, vol. 29, no. 3, pg. 493-502, 1975.

[16] P.-J. Wan, G. Călinescu, L.-W. Liu, O. Frieder: Grooming of Arbitrary Traffic in SONET/WDM Rings, to appear in *IEEE Journal of Selected Area on Communications.*

[17] P. Wilfong, P. Winkler, Ring Routing and Wavelength Translation, *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pg. 333-341, 1998.