Minimization of SONET ADMs in ring networks revisited

Leah Epstein* Asaf Levin † Betzalel Menahem ‡

Abstract

We design improved approximation algorithms for two variants of the ADM minimization problem. SONET add-drop multiplexers (ADMs) are the dominant cost factor in SONET/WDM rings. The number of SONET ADMs required by a set of traffic streams (lightpaths) in a ring is determined by the routing and the wavelength assignment of the traffic streams. We consider both the *arc version* where the route of each traffic stream is given as input, and the *chord version*, where the routing is to be decided by the algorithm. The goal in both cases is to assign wavelengths so as to minimize the total number of used SONET ADMs.

1 Introduction

The ADM minimization problem originates in Synchronous Optical Networks (SONET) and Wavelength Division Multiplexing (WDM) rings. In such architectures, each wavelength channel carries a high-speed SONET ring. The key terminating equipments are optical add-drop multiplexers (OADM) and SONET add-drop multiplexers (ADM). Each vertex is equipped with exactly one OADM. The OADM can selectively drop wavelengths at a vertex. Thus, if a wavelength does not carry any traffic from or to a vertex, its OADM allows that wavelength to optically bypass the vertex. Therefore, in each SONET ring, a SONET ADM is required at a vertex if and only if it carries some traffic terminating at this vertex. In this paper we study the problem of minimizing the total cost incurred by the SONET ADMs.

We next give a mathematical definition of the two problems, which are called the *arc version* and the *chord version*.

The arc version. We are given a set E of (directed) circular-arcs over the vertices $0, 1, \ldots, n-1$, where the vertices are ordered clockwise. The vertices form a ring in which every consecutive pair of vertices is connected by a physical link. An arc (i, j) represents the clockwise path along the ring from a vertex i to the vertex j. A pair of arcs $(i, j), (k, l)$ is *non-intersecting* if the clockwise path along the cycle $0, 1, \ldots, n-1, 0$, that connects i to j and the clockwise path that connects k to l, do not share any link of the ring. A set of arcs is non-intersecting if each pair of arcs from this set is non-intersecting. A feasible solution is a partition of E into non-intersecting subsets of arcs

*Department of Mathematics, University of Haifa, 31905 Haifa, Israel. lea@math.haifa.ac.il.

†Chaya Fellow. Faculty of Industrial Engineering and Management, The Technion, 32000 Haifa, Israel. levinas@ie.technion.ac.il.

‡Department of Mathematics, University of Haifa, 31905 Haifa, Israel.

$E_1, E_2, \ldots, E_p$. The cost of E_i is the number of different vertices of the ring that are endpoints of the arcs of E_i (where every endpoint counts only once, no matter whether it is an endpoint of just one arcs, or whether it is the endpoint of two arcs). The cost of the solution is the sum of costs of E_i for all i. The goal is to find a minimum cost feasible solution.

Liu, Li, Wan and Frieder [10] proved that this arc version of the ADM minimization problem is NP-hard. Several papers [10, 8, 13, 2, 12, 4] focused on the design and analysis of heuristics for the problem. Călinescu and Wan [2] provided a 3/2-approximation algorithm called PIM, which we describe later. Shalom and Zaks [12] obtained a $(\frac{10}{7} + \varepsilon)$-approximation algorithm. Their method is based on a preprocessing step that applies a local-search algorithm of [9] for finding an approximated maximum size sub-collection of 5-sets (sets with at most five elements). The current best result is a $98/69 \approx 1.42029$-approximation algorithm [4] (which was obtained independently of [12]). This last algorithm applies greedy steps and steps that find maximum weight matchings on graphs, but does not use local-search and thus it is very efficient in terms of running time. The problem on an arbitrary network (not necessarily a ring) was also studied before [3, 1, 6].

The chord version. This variant of the problem is defined as follows. We are given a set E of chords (or undirected edges) over the vertices $0, 1, \ldots, n-1$. The first goal is to decide on a routing (i.e. to convert each chord into an arc by choosing a single direction for it along the ring). This gives a set of arcs E', which becomes an input for the *arc version* of the problem, defined above. Călinescu and Wan [2] proved that this variant is NP-hard as well, and designed several heuristics, two of which are $\frac{3}{2}$-approximations. The current best algorithm, which has an approximation ratio of $\frac{10}{7}$, is due to [5].

Our results. In Section 2, we design an algorithm for the arc version which combines five different algorithms, and achieves an approximation ratio of at most $\frac{47}{34} + \varepsilon < 1.38236$. The chord version is considered in Section 3. For this version, we combine three algorithms on each connected component of the input to achieve an approximation ratio of at most $\frac{7}{5} + \varepsilon = 1.4 + \varepsilon$. The algorithms for the two problems present very different algorithmic methodologies. The first algorithm uses local-search for finding structures which can be given as output right away (these structures are then removed from the input), whereas the second algorithm incorporates enumeration in some cases in which the input contains small connected components.

2 Algorithms for the arc version

2.1 Preliminaries

For an arc (i, j), we define its *length* as $\ell(i, j) = j - i \mod n$. For a subset of arcs, the length of the subset is the total length of its arcs. Throughout the paper, the length of a path is defined to be the total length of its arcs.

A *chain* is an open directed path of length at most $n - 1$, and a *cycle* is a closed directed path of length exactly n. Without loss of generality, we can assume that the arcs in each E_i form a connected component (either a chain or a cycle). This is so because if the arcs in E_i are disconnected, then we can partition E_i to its connected components without increasing its total

cost. Therefore, we ask for a partition of E into cycles and (open) chains. Note that the cost of a feasible solution equals the sum of two terms, which are the value $|E|$ and the number of chains in the solution.

Let OPT be a given optimal solution to our problem with a cost which is denoted by OPT as well. Assume that OPT has CY_i cycles with i arcs, for $i = 2, 3, \ldots, n$. In addition, assume that OPT has CH_i chains with i arcs, for $i = 1, 2, \ldots, n-1$. We further assume that CY_2 is maximized among all optimal solutions. For an algorithm A, we also use A to denote the cost of its returned solution.

We use the following auxiliary definition. *The deficiency of a vertex* v, $def(v)$, is defined as follows. Let $in(v)$ be the number of ingoing arcs of v, and let $out(v)$ be the number of outgoing arcs of v. Then, $def(v) = \frac{1}{2}|in(v) - out(v)|$.

In what follows, Eulerian subgraphs are discussed. Our notion of a subgraph being Eulerian allows the subgraph to be disconnected. Thus, by an Eulerian subgraph we mean a union of Eulerian connected subgraphs, which form a possibly disconnected graph. A feasible solution SOL induces a partition of the arcs into an Eulerian subgraph and a set of *mega-chains* as follows: We consider the set of cycles and chains used by SOL as a set of arcs in directed auxiliary graph over $\{0, 1, \ldots, n-1\}$ where cycles are loops of the form $(0, 0)$, and a chain is a directed arc from its starting vertex to its end vertex. In this directed graph we find a maximal subgraph in which the in-degree of each vertex equals its out-degree. The remaining arcs define a minimal set of paths, such that each such path is directed from a vertex whose out-degree is greater than its in-degree, towards a vertex whose in-degree is greater than its out-degree. Each such path in the auxiliary graph corresponds to a *mega-chain* in the original graph (by replacing each arc in the auxiliary graph by its corresponding chain). Therefore, each mega-chain is not necessarily a valid chain,

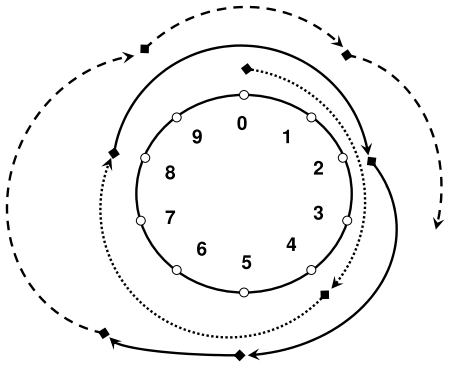

Figure 1: An example of a single mega-chain which consists of three chains. The mega-chain starts at vertex 0 and ends at vertex 3. The first chain contains the arcs $(0, 4)$, $(4, 8)$, the second chain contains the arcs $(8, 2)$, $(2, 5)$, $(5, 6)$, and the third chain contains the arcs $(6, 9)$, $(9, 1)$ and $(1, 3)$. The value of n is 10.

but it can be decomposed into a sequence of valid chains. Note that the number of mega-chains in SOL is equal to the total deficiency of all vertices. Therefore it is independent of SOL, and is common to all feasible solutions. See Figure 1 for an example of a set of arcs E which forms a single mega-chain.

It was noted in [2] that a greedy removal of cycles which consist of two arcs, before the application of some algorithm can never degrade the performance, since there always exists an optimal solution which contains all such cycles as components. Therefore, each algorithm, which we consider, contains a preprocessing phase in which such two-arc cycles are removed from the input. By the choice of OPT, the value CY_2 is the maximum possible number of two-arc cycles which any algorithm can produce.

2.1.1 Algorithm PIM

We first define the algorithm Iterative Matching (IM), which is used as a subroutine in Algorithm PIM [2]. The algorithm maintains a set of valid chains of arcs $\mathcal{P}$, which forms a partition of E, throughout its execution. Initially, $\mathcal{P}$ consists of chains each of which is an arc in E. Given a partition $\mathcal{P}$, the fit graph $\mathcal{F}(\mathcal{P})$ is defined as follows. Its vertex set is $\mathcal{P}$, and two of its vertices are connected by an edge if the two corresponding chains have a common endpoint, and they can be concatenated to form a valid chain (or a cycle). The algorithm constructs $\mathcal{F}(\mathcal{P})$, and if its edge set is not empty, then it finds a maximum matching $\mathcal{M}$ in $\mathcal{F}(\mathcal{P})$. Then, it merges each matched pair of chains of arcs in $\mathcal{M}$ into a longer chain. When the edge set of $\mathcal{F}(\mathcal{P})$ is empty, $\mathcal{P}$ is the valid solution which is given as output. It was shown in [2] that the approximation ratio of Algorithm IM is at most $\frac{5}{3}$ and at least $\frac{3}{2}$. The lower bound was later improved to 1.6 and to $\frac{14}{9}$ if two-arc cycles are removed before the application of IM [4]. Finally, a tight upper bound of 1.6 (for the variant without any preprocessing) was proved by Flammini, Shalom and Zaks [7].

The main drawback of IM is that it gives no advantage to cycles over open chains. In a cycle, the number of ADM's is equal to the number of arcs, and hence it is better to have cycles rather than chains. The following algorithm Preprocessed Iterative Matching (PIM) [2] was defined in order to exploit the advantage of cycles over chains.

1. Preprocessing phase: repeatedly remove cycles consisting of remaining arcs until no more cycles can be obtained. Give preference to cycles consisting of two arcs.

2. Matching phase: apply Algorithm IM to the arcs remaining after the first phase.

The approximation ratio of this algorithm is $\frac{3}{2}$, where the upper bound follows from [2] and the lower bound from [4]. A careful analysis of PIM, which resembles the proof of Lemma 19 in [2] and the proof of Theorem 2 in [4], results in the following theorem (we present its proof for completeness).

Theorem 1 *Algorithm* PIM *returns a solution whose cost is at most* $2CY_2 + \sum_{i=3}^{n} \lfloor \frac{3i}{2} \rfloor CY_i + \sum_{i=1}^{n-1} \lfloor \frac{3i+1}{2} \rfloor CH_i$.

Proof. Note that cycles are removed greedily and so OPT may have a very different set of cycles. However, PIM moves to the second phase only after no cycles remain, and therefore, every cycle of OPT has at least one arc which is chosen as a part of a cycle of PIM.

Instead of considering the solution obtained by PIM, we analyze an alternative solution whose cost is at least the cost of PIM. To be precise, we replace the solution of PIM with the solution obtained by PIM after just one iteration of the matching phase. The cost of this solution is clearly an upper bound on the cost of PIM. We further replace the solution by a possibly inferior solution that does not create a maximum cardinality matching, but a feasible matching which we construct. This matching is constructed by uniting matchings that may be created from the remaining arcs (after the preprocessing phase, performed by PIM), taken from each component of an optimal solution OPT (cycle or chain) separately.

Consider a cycle of OPT, which consists of $i > 2$ arcs. Note that the preprocessing phase of PIM removes at least one arc of this cycle to participate in some cycle created by PIM. We consider the arcs remaining after the removal of exactly one arc (the first arc ever removed) and partition them into consecutive pairs of arcs. If i is even, then one arc remains unpaired. For each pair of arcs, if none of them is removed in the preprocessing, we match them in the matching which we define. All other arcs are not matched. We perform a similar process on chains of any number of arcs, only we do not remove any arcs before applying the pairing process (even if some arcs were removed). For a cycle where $i = 2$, we note that by the definition of PIM, both its arcs were removed together as a cycle in the preprocessing phase.

We next analyze the cost of the resulting solution. The cost of every two-arc cycle of OPT is 2. We next consider the cost of a cycle consisting of $i > 2$ arcs. The cost of each pair of arcs is at most 3. Specifically, this is the cost if this pair of arcs is defined to be a part of the matching, or if exactly one of the arcs of the pair was removed in the preprocessing. If both arcs of a pair were removed in the preprocessing, then the cost of the pair of arcs is 2. If i is odd, then adding the removed arc, the total cost of the cycle is $\frac{3(i-1)}{2} + 1 = \frac{3i-1}{2}$, while for even i, we need to consider the unpaired arc, and we get a total of at most $\frac{3(i-2)}{2} + 2 + 1 = \frac{3i}{2}$. A similar calculation for chains gives a cost of at most $\frac{3(i-1)}{2} + 2 = \frac{3i+1}{2}$ for odd i and of at most $\frac{3i}{2}$ for even i. Thus the cost for a cycle of $i > 2$ arcs is $\lfloor \frac{3i}{2} \rfloor$ and the cost for a chain of i arcs is $\lfloor \frac{3i+1}{2} \rfloor$, as claimed above. ■

2.2 New algorithms

We design an algorithm MANY, which chooses the best solution out of the solutions produced by five algorithms. The first algorithm used by MANY is the variant of PIM above (which starts with an optimal removal of two-arc cycles). The other four algorithms R_k ($k = 1, 2, 3, 4$) combine techniques from both [4] and [12]. Previously designed algorithms, which can be found in [2, 3] apply just one phase of processing on the entire input. Their relatively high approximation ratios are caused by examples, where an optimal solution contains a large number of components of a given type. A natural approach, which is used by both [4] and [12] is to apply a preprocessing step where the algorithm is trying to construct such problematic components before applying any general heuristic. In [4], a number of preprocessing steps is used, in which cycles and mega-chains are

greedily detected to be removed from the input and are given as output. In [12], the preprocessing is applied in a single step, which applies an algorithm of [9] (see below) to detect short cycles. The advantage of the greedy preprocessing steps is that the analysis method developed in [4] allows to apply several such steps, sequentially. The advantage of using the algorithm of [9] is that it allows to remove a larger number of components in the preprocessing step. In this paper, we develop a method which allows to combine the two options. We apply both a greedy preprocessing step, and a step which uses the algorithm of [9]. The additional novelty is that we use a number of algorithms with slightly distinct preprocessing phases, in order to deal with multiple problematic inputs.

Each of the algorithms, on which MANY is based, applies a number of preprocessing phases, in which we remove some set of subgraphs, and afterwards we apply the final step for the remaining arcs in which a set of mega-chains and an Eulerian subgraph are computed in an arbitrary way, and each one of them is traversed and partitioned greedily. To do so, we apply (similarly to [4]) a variant of an algorithm studied in [2], called EULERIAN TOUR-TRAIL SPLITTING, which partitions the remaining arcs into valid chains. In the analysis of these algorithms we allocate costs to arcs. The cost of an arc consists of a unit cost, which is the minimum cost of ADM's for an arc, and in addition, for each $\frac{n}{2}$ units of length (of the arcs remained for the final step) we will pay one additional unit. We will also pay 1 for each mega-chain. Hence, the performance guarantee of the final step for components which are either a cycle of a small number of arcs, or a mega-chain of a small number of arcs but with a large length, is not good enough. This is exactly the role of the preprocessing step in the analysis. However, applying just one type of preprocessing is beneficial for just one type of problematic input. Thus, in each algorithm, we perform a different such step whose aim is to remove subset of these components. We note that we can remove components only approximately if the number of elements in the component is at least 3, no matter how this step is implemented. The approximation ratio of this removal phase, that is, the fraction of required components which are successfully found, is thus a major factor in our selection of algorithms. We formulated the resulting approximation ratio if we incorporate any subset of these algorithms, and each time we identified the bottleneck set of components, that is, the type of cycles or chains which are the worst case example for the algorithm. In the design of the next algorithm to be added, these components are the ones to be then removed in the preprocessing. This process is continued until we can no longer improve the resulting approximation ratio by using additional algorithms. We next present the resulting set of algorithms, and note that the constants were selected so as to optimize the resulting approximation ratio (see details below).

Notations. We use the following notations for components of OPT. We consider mega-chains of OPT neglecting their partition into valid chains. The number of mega-chains in the optimal solution, which consist of a single arc, is denoted by MC_1, and the total length of these mega-chains is denoted by L_1. The number of mega-chains in the optimal solution which have at least four arcs, and length in $[(i-1)n, in-1]$, is denoted by MC^i. The mega-chains of two and three arcs are partitioned into further categories as follows. The partition is due to a different treatment in some of the algorithms.

MC_2^1 is the number of two-arc mega-chains with length in $[1, n-1]$. MC_2^2 is the number of two-arc mega-chains with length in $[n+1, 31n/18)$. MC_2^3 is the number of two-arc mega-

chains with length in $[31n/18, 7n/4)$. MC_2^4 is the number of two-arc mega-chains with length in $[7n/4, 11n/6)$. MC_2^5 is the number of two-arc mega-chains with length in $[11n/6, 2n-1]$. Finally, we let $MC_2 = \sum_{j=1}^{5} MC_2^j$ be the total number of two-arc mega-chains.

MC_3^1 is the number of three-arc mega-chains with length in $[1, 3n/4)$. MC_3^2 is the number of three-arc mega-chains with length in $[3n/4, n-1]$. MC_3^3 is the number of three-arc mega-chains with length in $[n+1, 2n-1]$. MC_3^4 is the number of three-arc mega-chains with length in $[2n, 3n-1]$. Finally, we let $MC_3 = \sum_{j=1}^{4} MC_3^j$ be the number of three-arc mega-chains.

We use MC to denote the total number of mega-chains, and $CY = \sum_{i=7}^{n} CY_i$ to denote the number of cycles with at least seven arcs. The number of chains which belong to the Eulerian subgraph and consist of i arcs is denoted by CH_E^i.

Building blocks of the algorithms. Each one of the algorithms (R_1, R_2, R_3, R_4) consists of at most five steps as follows which are the common building blocks of all these algorithms.

The first step is a removal of a maximum number of two-arc cycles. This step can be performed greedily and yields optimal results.

For the second step, it is required to know the exact value of MC_1. This value is unknown to the algorithm, but must take an integer value in $[0, |E|]$. To overcome the lack of knowledge, each algorithm is executed $|E|+1$ times and the best solution (that is, the solution which uses the least number of ADMs) is reported as output. We analyze the solution which uses the correct value of MC_1. The algorithm removes (if possible) exactly MC_1 mega-chains of one arc, of maximum total length. This length is at least L_1 for the correct value of MC_1. This step can be applied using an algorithm for a maximum weight b-matching on a bipartite graph, whose parts are the vertices with non-zero deficiency (see [4] for details of this reduction to maximum b-matching). In order to achieve a good performance, mega-chains of one arc must receive this special treatment. These are components of a very large cost per arc, and thus leaving even a small fraction of them for later results in a very high cost per the single arc of which the mega-chain consists. Even though this step does not necessarily remove exactly the set of such mega-chains of Opt, still, if the correct value of MC_1 is used, it removes a total length of such chains which is sufficiently large.

The third step is a greedy preprocessing step. In this step the algorithm seeks to create structures of a given type (which could be cycles or chains, but in our case these are mega-chains of a given number of arcs and length in a given interval). This step is performed greedily. By greedy we mean that the algorithm maintains a sub-collection of disjoint structures of the given type, and as long as this sub-collection can be extended by a disjoint structure (of the given type), we add to the sub-collection such a structure. It turns out that the components causing high approximation ratios of previously known algorithms are cycles and mega-chains with few arcs. These are the structures which receive most attention in this step and the next step.

The fourth step is a local-search step which is performed using a local-search method which was analyzed by Hurkens and Schrijver [9]. This heuristic receives as an input a collection S of valid subsets of a given ground set, each containing at most k elements, and finds a sub-collection

S' of S such that the intersection of every two members of S' is empty, and $|S'|$ is at least $\frac{2}{k} - \varepsilon$ of the size of a maximum cardinality sub-collection of disjoint sets. The running time is polynomial in the input size (though it is exponential in $\frac{1}{\varepsilon}$). In our case, the subsets are either valid cycles or mega-chains. For a cycle, the subset contains all its arcs. For a mega-chain, the subset contains its arcs as well as its endpoints, since the removal of the arcs of a mega-chain from the input reduces the deficiency of its two endpoints. Note that we are dealing with multigraphs, i.e., an arc can exist multiple times. In addition, endpoints of mega-chains may exist multiple times, but a given vertex can either be a starting vertex of some mega-chain, or the last vertex of some mega-chain, but not both. We therefore treat multiple instances of arcs or endpoints as distinct elements.

Note that in both last steps, we require that a removed mega-chain is be partitioned into valid chains in an optimal way before it becomes part of the output. In these steps, we never remove mega-chains with more than three arcs, and thus the calculation of an optimal partition requires constant time per mega-chain.

The fifth step is the creation of chains out of the remaining arcs. First, a set of mega-chains and an Eulerian subgraph are computed in an arbitrary way. Afterwards, each one of them is traversed and partitioned greedily. At each time, a maximum length valid chain is chosen to be a part of the output.

Statement of the algorithms. The first algorithm, R_1 applies steps 1,2,4 and 5, and does not have a third step. In the fourth step, it removes valid cycles which consist of three or four arcs.

The algorithm R_2 applies all five steps. In step 3 it performs a greedy removal of two-arc mega-chains of length in $[\frac{11n}{6}, 2n-1]$, and the fourth step is identical to the fourth step of R_1, that is, it removes valid cycles which consist of three or four arcs. The exact choice of the length of "long" mega-chains is done in order to balance the cost of a shorter mega-chain (which is not considered to be removed) with a longer one. A mega-chain which has a length shorter than $\frac{11n}{6}$ only adds an additive factor of at most $\frac{11}{3}$ in the cost which is based on the last phase (rather than a cost of almost 4, caused by a mega-chain of length close to $2n$). On the other hand, a mega-chain of length at least $\frac{11n}{6}$ which is removed in the preprocessing, is exempt of the cost of the last step, and thus, a cost of at least $\frac{11}{3}$ is saved. Note that increasing the value $\frac{11n}{6}$ would increase the maximum cost of a short mega-chain as well as increasing the saved cost of a long mega-chain.

The algorithm R_3 also applies all five steps, but its exact action is different. In step 3 it applies a greedy removal of two-arc mega-chains of length in $[\frac{31n}{18}, 2n-1]$, and in the fourth step, cycles of at least three and at most six arcs are removed.

The algorithm R_4 applies steps 1,2,4 and 5. It does not apply a greedy step (the third step). In step 4 it removes the following structures at once. Cycles of at most six arcs (and at least three arcs), mega-chains of two arcs and length in $[\frac{7n}{4}, 2n-1]$, and mega-chains of three arcs and length in $[\frac{3n}{4}, n-1]$.

Analysis. For each algorithm, we distribute its cost among the different components of Opt. This allows us to consider the approximation ratio on each component separately. We calculate the cost charged to a set of chains, which result from one mega-chain, together. As a first step, we calculate the cost incurred by each component of Opt. These costs are described in the last column of Table 1. All these costs are straightforward except for the costs of mega-chains of at

least two arcs. The cost for a mega-chain of length at most $n-1$ is simply the number of arcs plus 1, since such a mega-chain does not need to be split into chains. On the other hand, the cost of a mega-chain can never exceed twice the number of arcs, which corresponds to the option of partitioning it into one-arc chains. For a mega-chain of three arcs which must be split (i.e., of length of at least $n+1$), it needs to either be split into two or three chains, and we consider these two cases separately. As for a mega-chain of $t \geq 4$ arcs and length in $[(i-1)n, in-1]$, the number of chains into which it must be split is at least i.

We use the following upper bound UB_L on the total length of all arcs in the input. We later compute an upper bound on the residual cost remaining after the execution of the first four steps of each algorithm (either explicitly, or by calculating the reduction in length compared to UB_L).

$$UB_L = (\sum_{j=2}^{6} CY_j + CY)n + L_1 + \sum_{i=1}^{\infty}(CH_E^i \cdot n) + MC_2^1 \cdot n + MC_2^2 \cdot \frac{31n}{18} + MC_2^3 \cdot \frac{7n}{4} \qquad (1)$$
$$+MC_2^4 \cdot \frac{11n}{6} + MC_2^5 \cdot 2n + MC_3^1 \cdot \frac{3n}{4} + MC_3^2 \cdot n + MC_3^3 \cdot 2n + MC_3^4 \cdot 3n + \sum_{i=1}^{\infty}(MC^i \cdot in) \ .$$

Next, we explain how to distribute the cost of an algorithm among the components of Opt. We define a basic cost for a structure of Opt, based on an algorithm which does not apply the third and fourth steps. Later we extend the costs for our algorithms which apply these steps as well.

We note that the number of chains created in the last step is at most the sum of the following. The number of mega-chains, and the total length of the arcs, excluding the arcs removed in steps 1 and 2, divided by $\frac{n}{2}$. It was shown in [4] that the average length of a chain which is created from the Eulerian part is more than $\frac{n}{2}$, whereas a mega-chain of length in $[kn, (k+1)n-1]$ creates at most $2k+1$ chains. Thus if at the beginning of step 5 the total length of remaining arcs is T, and the number of remaining mega-chains (after MC_1 mega-chains were removed in step 2) is P, then the number of chains created in the fifth step is at most $\frac{2T}{n} + P$.

The reduction in the total length of all arcs, as a result of the application of the first step and the second step, is at least $CY_2 \cdot n + L_1$, thus, the number of chains created by the algorithm is at most $MC_1 + \frac{UB_L - CY_2 \cdot n - L_1}{n/2} + P$, where P is the number of mega-chains, remaining after step 2. The cost of the solution is at most this value plus the number of arcs. Thus, the cost assigned to each structure of Opt is the number of arcs in it, plus some amount which results from the number of chains.

The cost assigned to a two-arc cycle is 2, as in Opt, and the cost assigned to a one-arc mega-chain is 2 as well, since it contains one arc, and contributes 1 to the number of mega-chains. The cost assigned to the other components takes into account the cost of chains created in the fifth step. In addition to a cost of 1 for every arc, and another cost of 1 for each mega-chain, we need to distribute a cost of 1 among the components of Opt for each other chain which is created. Given a component of Opt which is not a two-arc cycle, or one-arc mega-chain, we assign these costs according to the coefficient of the variable corresponding to the number of such components in Opt in $\frac{UB_L}{n/2}$. Thus, for example, a two-arc mega-chain of Opt, with length in $[n+1, 31n/18)$ is assigned

an additional cost of $\frac{31}{9}$, in addition to the cost of 3 resulting from its arcs, and from it being a mega-chain.

In the sequel when we analyze the different algorithms, we neglect the ε and assume that a fraction of $\frac{2}{k}$ (instead of $\frac{2}{k} - \varepsilon$) is found during step 4. In order to get a correct upper bound a small constant (linear in ε) must be added to the approximation ratio.

Analysis of algorithm R_1. Consider the first algorithm, R_1. In the fourth step, it removes valid cycles which consist of three or four arcs. Using [9], we find that a fraction of at least $\frac{1}{2} - \varepsilon$ of a maximum cardinality set of arc disjoint cycles is found. Let C and D denote the number of removed cycles of three and four arcs, respectively. A lower bound on the total length of the structures removed in the first two steps and the local-search step implies a reduction in the total length of arcs compared to (1) as follows: $Cn + Dn + L_1 + CY_2 n$. The maximum number of cycles which can be removed simultaneously in the fourth step is at least $CY_3 + CY_4 - MC_1$, since the removal of mega-chains can be harmful to cycles of the optimal solution. Neglecting the ε, we get the inequality $C + D \geq \frac{CY_3 + CY_4 - MC_1}{2}$. Thus the reduction in length is therefore at least $\frac{CY_3 n}{2} + \frac{CY_4 n}{2} - \frac{MC_1 n}{2} + L_1 + CY_2 n$.

The total length of arcs remaining for step 5 is no larger than

$$\frac{CY_3 \cdot n}{2} + \frac{CY_4 \cdot n}{2} + CY_5 \cdot n + CY_6 \cdot n + CY \cdot n + \frac{MC_1 \cdot n}{2} + \sum_{i=1}^{\infty}(CH_E^i \cdot n) + MC_2^1 \cdot n$$
$$+(\sum_{j=2}^{5} MC_2^j) \cdot 2n + (MC_3^1 + MC_3^2) \cdot n + MC_3^3 \cdot 2n + MC_3^4 \cdot 3n + \sum_{i=1}^{\infty}(MC^i \cdot in).$$

We allocate the cost of the solution analogously to the basic case, as follows. Every arc is charged with 1. In addition, each mega-chain causes an additional cost of 1. Finally, the total length of arcs which are assigned to chains in step 5 causes a cost of 1 for a length of $\frac{n}{2}$. Hence, by using the upper bound on the total length of arcs remaining for step 5, we allocate the last term to the different structures of Opt such that if in this upper bound we have $\beta \cdot \frac{n}{2}$ times the number of this structure in Opt, then we assign this structure an additional cost of β. Assigning costs to structures of Opt using this rule results in the costs of R_1 in Table 1.

Analysis of algorithms R_2 and R_3. We next analyze the two algorithms R_2 and R_3 together. The fraction of the cycles which algorithm R_3 removes in step 4 (out of the maximum feasible set that can be removed simultaneously) is at least $\frac{1}{3}$, while for R_2 it is $\frac{1}{2}$. The outline of the analysis is as follows. We first consider the basic cost of each structure in Opt. This basic cost is defined by allocating a unit cost for each arc, an additional unit cost for each mega-chain, and another unit cost for each $\frac{n}{2}$ units of lengths. Then, we use the combinatorial structure of steps 2,3 and 4 to conclude that this basic cost has decreased in a sufficient way such that the resulting allocation cost is as in the table. The cost decreases due to a decrease in the total length of arcs, but on the other hand, there is an increase which is caused by the fact that the removed mega-chain is invalid. We next describe this in detail.

We use the following notations. Let α ($\alpha = \frac{11}{6}$ or $\alpha = \frac{31}{18}$) be such that the mega-chains of

two-arcs which are removed in step 3 must have length in $[\alpha \cdot n, 2n-1]$. Let k be the maximum number of arcs in cycles which can be removed in step 4 ($k = 4$ or $k = 6$).

In order to find how many structures remain to be considered in step 4, after some structures were removed in steps 2 and 3, we use some additional notations. Let MC_1^C denote the number of arcs removed in the second step which belong to cycles of the optimal solution, and let $MC_1^M = MC_1 - MC_1^C$ denote the number of other removed arcs, which is an upper bound on the number of arcs of the removed mega-chains in step 2 which are part of mega-chains of OPT. We denote the number of cycles of j arcs removed in the fourth step by C_j, and the number of mega-chains removed in the third step by B. Out of these arcs, we let B^C be the number of arcs of the removed mega-chains which belong to cycles in OPT, and $B^M = 2B - B^C$. Let K denote the number of two-arc mega-chains of length at least αn in OPT, and let J denote the number of cycles of at least three and at most k arcs in OPT.

We use these notations to state the fact that the third step stops only when no two-arc mega-chains of sufficient length remain. At this time, at least one endpoint or arc was removed from each such mega-chain of OPT. Thus $K \leq 2B + B^M + 2MC_1 + MC_1^M = 4B - B^C + 3MC_1 - MC_1^C$. On the other hand, arcs of removed mega-chains affect the fourth step only if they are arcs of cycles of OPT. Thus $\sum_{j=3}^{k} C_j \geq \frac{J - B^C - MC_1^C}{k/2}$. Multiplying the first inequality by $0 < \gamma \leq 1$ and adding the resulting inequality to $\frac{k}{2}$ times the second one, we get $\frac{k}{2}(\sum_{j=3}^{k} C_j) + \gamma(4B - B^C + 3MC_1 - MC_1^C) \geq \gamma K + J - B^C - MC_1^C$ or using $B^C \leq 2B$, $MC_1^C \leq MC_1$ and $\gamma < 1$, we conclude that $\frac{k}{2}(\sum_{j=3}^{k} C_j) + (2 + 2\gamma)B + (2\gamma + 1)MC_1 \geq \gamma K + J$.

The upper bound on the total length is reduced during steps 2,3 and 4 by $nCY_2 + L_1 + (\sum_{j=3}^{k} C_j)n + B\alpha n \geq nCY_2 + L_1 + \frac{2\gamma}{k}Kn + \frac{2}{k}Jn - \frac{2(2\gamma+1)}{k}MC_1 n + B(\alpha n - \frac{4(\gamma+1)n}{k})$, thus due to the total reduced length, the cost of the algorithm in the last step is reduced by $2CY_2 + \frac{2}{n}L_1 + \frac{4\gamma}{k}K + \frac{4}{k}J - \frac{4(2\gamma+1)}{k}MC_1 + 2B(\alpha - \frac{4(\gamma+1)}{k})$. However the cost increases by B since the cost of a mega-chain of two arcs of a length which exceeds n is 4, and not 3, as the charge for it, excluding the charge for the length as defined in the basic costs. Hence the total reduction of the cost (with respect to the basic cost) is at least $2CY_2 + \frac{2}{n}L_1 + \frac{4\gamma}{k}K + \frac{4}{k}J - \frac{4(2\gamma+1)}{k}MC_1 + 2B(\alpha - \frac{4(\gamma+1)}{k} - \frac{1}{2})$. We now characterize this bound for each of the algorithms R_2 and R_3.

For the case of R_2, we have $k = 4$, $\gamma = \frac{1}{3}$, $\alpha = \frac{11}{6}$, $K = MC_2^5$ and $J = CY_3 + CY_4$. We conclude that the cost decreases by at least $2CY_2 + \frac{2}{n}L_1 + \frac{1}{3}MC_2^5 + CY_3 + CY_4 - \frac{5}{3}MC_1$. This implies the costs in the table.

For the case of R_3, we have $k = 6$, $\gamma = \frac{5}{6}$, $\alpha = \frac{31}{18}$, $K = MC_2^3 + MC_2^4 + MC_2^5$ and $J = CY_3 + CY_4 + CY_5 + CY_6$. We conclude that the cost decreases by at least $2CY_2 + \frac{2}{n}L_1 + \frac{5}{9}(MC_2^3 + MC_2^4 + MC_2^5) + \frac{2}{3}CY_3 + \frac{2}{3}CY_4 + \frac{2}{3}CY_5 + \frac{2}{3}CY_6 - \frac{16}{9}MC_1$. This implies the costs in the table.

Analysis of algorithm R_4. To analyze R_4, we use the following notations. Denote by X the number of two-arc mega-chains with length in $[\frac{7n}{4}, 2n-1]$ which we removed during step 4. Denote by Y the number of three-arc mega-chains and length in $[\frac{3n}{4}, n-1]$ which we removed during step 4. Denote by Z the number of cycles of at most six arcs (and at least three arcs) which we removed during step 4. Since each one-arc mega-chain which we removed in step 2 may intersect at most three structures of OPT, we conclude by the performance guarantee of the local-search procedure that $X + Y + Z \geq \frac{CY_3 + CY_4 + CY_5 + CY_6 + MC_2^4 + MC_2^5 + MC_3^2 - 3MC_1}{3}$. Note that the total

length reduction resulting from applying step 4 is at least $\frac{7}{4}Xn + \frac{3}{4}Yn + Zn$. Since each two-arc mega-chain which we removed costs 4 and not 3, as the charge for it excluding the charge for the length as defined in the basic costs, the total decrease of cost due to applying step 4 is at least $\frac{7}{2}X + \frac{3}{2}Y + 2Z - X \geq \frac{3}{2}(X+Y+Z) \geq \frac{CY_3+CY_4+CY_5+CY_6+MC_2^4+MC_2^5+MC_3^2-3MC_1}{2}$. This implies the costs in the table.

Bounding the cost of Many. We conclude the entries in the table by the following theorem.

Theorem 2 *The approximation ratio of* Many *is at most* $\frac{47}{34} + \varepsilon$.

Proof. Since Many chooses the best solution out of the five outputs, its cost is no larger than any convex combination of the costs. For each structure S of Opt mentioned in Table 1 (that is, a row in the table), we denote the cost of an algorithm A on S (as we define it above, for each one of the algorithms) by $\mathrm{A}(S)$. We compute an upper bound on $\frac{1}{34}(15\textsc{Pim}(S)+9R_1(S)+2R_2(S)+6R_3(S)+2R_4(S))$. It suffices to show that this upper bound is at most 47 times the cost of Opt on S. For most structures S, the result follows from the column $34 \cdot \textsc{Many}$, which contains the numerical value of $15\textsc{Pim}(T) + 9R_1(T) + 2R_2(T) + 6R_3(T) + 2R_4(T)$, and then by computing $47\textsc{Opt}(S)$ one can verify the correctness of the claim for these structures. Among these note that for a mega-chain of three arcs and length in $[n+1, 2n-1]$, if $\textsc{Opt} = 5$, it means that there exists a pair of arcs in this mega-chain that can form a chain together, and therefore, Pim can also create such a chain and we charge it with the same cost.

It remains to consider the rows for which the entry of $34 \cdot \textsc{Many}(S)$ is non-numerical (and depends on some parameter). For these structures we prove in the sequel the claimed ratio.

For a cycle of $i \geq 7$ arcs, we have $\frac{\textsc{Many}(T)}{\textsc{Opt}} \leq \frac{41.5i+38}{34i} \leq \frac{41.5+38/7}{34} < \frac{47}{34}$.

For a mega-chain of $t \geq 4$ arcs, note that the cost of Pim follows from a cost of $\frac{3j+1}{2}$ for a chain of j arcs. Thus the cost is $\frac{3}{2}$ times the number of arcs, plus $\frac{1}{2}$ times the number of chains. Using $t \geq 4$, we have $7.5k + 38i + 41.5t + 19 \leq 7.5k + 38i + 46.5t \leq 7.5(t+k) + 39(t+i) < 47 \cdot \textsc{Opt}$.

For chains of the Eulerian subgraph, we have $\frac{\textsc{Many}(S)}{\textsc{Opt}} \leq \frac{41.5i+45.5}{34(i+1)} \leq \frac{45.5}{34} < \frac{47}{34}$. ■

3 The chord version

In this section, we assume that the graph of chords is connected. Otherwise, the algorithm should be applied on every connected component separately. Since any solution must consider each connected component separately, the outputs for different connected components are independent, and an upper bound on the approximation ratio of the algorithm for connected graphs implies the same approximation ratio for disconnected graphs.

We use the same definitions as for the arc version for dealing with the chord version. Note that for this problem, the removal of two-chord cycles is not optimal and can harm the performance of algorithms [2, 5].

The algorithms IM and Pim above can be converted into algorithms for the chord version. IM can act on chords instead of arcs, where chords are assigned a routing only when they are matched

The structure of Opt	Pim	R_1	R_2	R_3	R_4	$34 \cdot$ Many	Opt
A two-arc cycle	2	2	2	2	2	68	2
A three arc cycle	4	4	4	13/3	4.5	139	3
A four arc cycle	6	5	5	16/3	5.5	188	4
A five arc cycle	7	7	7	19/3	6.5	233	5
A six arc cycle	9	8	8	22/3	7.5	282	6
A cycle of $i \geq 7$ arcs	$3i/2$	$i+2$	$i+2$	$i+2$	$i+2$	$\frac{83i}{2}+38$	i
A mega-chain of a single arc	2	3	11/3	34/9	3.5	94	2
Mega-chains of two arcs:							
length in $[1, n-1]$	3	5	5	5	5	140	3
length in $[n+1, 2n-1]$	4	7	20/3	58/9	6.5	188	4
Mega-chains of three arcs:							
length in $[1, n-1]$	5	6	6	6	5.5	188	4
length in $[n+1, 2n-1]$, Opt $= 5$	5	8	8	8	8	227	5
length in $[n+1, 3n-1]$, Opt $= 6$	6	10	10	10	10	280	6
A mega-chain, $t \geq 4$ arcs							
length in $[(i-1)n, in-1]$	$k/2+$	$2i+$	$2i+$	$2i+$	$2i+$	$\frac{15k}{2}+38i$	$t+k \geq$
which Opt splits into k chains	$3t/2$	$t+1$	$t+1$	$t+1$	$t+1$	$+\frac{83t}{2}+19$	$t+i$
A chain of Opt with i arcs							
(a part of the Eulerian subgraph)	$\frac{3i+1}{2}$	$i+2$	$i+2$	$i+2$	$i+2$	$\frac{83i}{2}+\frac{91}{2}$	$i+1$

Table 1: An analysis of Many

to another chord or to an existing chain. Pim can remove cycles allowing every chord to be routed in one of its possible directions.

To motivate our improved algorithm, consider first a special case in which the (connected) input does not contain any cycles. Note that a pair of edges which have a common endpoint can always be routed in a way that a two-arc chain can be created out of them. Therefore, in this case, we are guaranteed that the solution returned by Pim (or IM) has at most a single chain consisting of a single chord, while all other chains contain at least two edges each. This unique chain causes the analysis of the algorithm in [5] to be loose. Note that this one chain can be discarded if the input is large (i.e., if $|E| = \omega(\frac{1}{\varepsilon})$, then this chord increases the cost of the solution by at most $O(\varepsilon)$Opt). Therefore, in this case we would like to restrict ourselves to inputs in which the number of chords is large. Such a restriction is possible, if we solve other (small) inputs optimally (via exhaustive enumeration). The same situation holds also if the input contains cycles, but in this case, the accounting of the number of single chord chains should be made more carefully. Specifically, the removal of a cycle may disconnect the remaining set of edges into some connected components, but the removal of a cycle is sufficiently beneficial, so that it can compensate for the creation of additional chains consisting of a single chord.

For an input consisting of large number of chords, the algorithm which we apply is similar to

the one of [5]. That algorithm combines a pair of algorithms, DDAG and a variant of PIM, choosing the solution of the smaller cost as an output. DDAG routes all chords in a specific way, creating an acyclic input for the arc version. This destroys all potential cycles, and it works relatively well for inputs where OPT has many chords in most components, since just one edge of a component is routed in the opposite direction compared to the routing in OPT. The second algorithm is the variant of PIM for the chord problem, in which cycles of two edges are considered *last* (here we do not give preference to cycles of a larger number of edges). It works well on the complement set of the instances, since due to its greediness, it works well on components with a small number of edges. The improved bound on the cost of PIM allows us to prove a tighter approximation ratio. For inputs consisting of a small number of chords, we solve the problem optimally, and hence the approximation ratio in this case is proven trivially.

More accurately, algorithm DDAG acts as follows. It chooses an arbitrary link of the ring, and orients all chords so that they do not contain this link. As a result, the collection of arcs is acyclic, and an optimal algorithm for this set of arcs, called Greedy Sweeping, due to [8], is applied.

The following lemma is proved in [5].

Lemma 3 *The cost of* DDAG *is at most*

$$\sum_{i=1}^{n}(i+3)CH_i + \sum_{i=2}^{n}(i+2)CY_i.$$

Let $\varepsilon > 0$. We define an algorithm COMBENUM, for which we prove an approximation ratio of $\frac{7}{5} + \varepsilon$. If the graph contains at most $\frac{2}{5\varepsilon}$ chords, the algorithm returns an optimal solution by enumerating all possible solutions. Each potential solution is a partition of edges into subsets. A subset is valid if it implies a single cycle or chain.

Otherwise, COMBENUM returns the best solution out of the two solutions returned by DDAG and PIM. Note that the variant of PIM used here is the algorithm which removes valid cycles of arbitrary length from the input until no such cycles remain (i.e., unlike the algorithm in [5], we do not delay the removal of two-arc cycles until there are no other cycles), and applies IM afterwards.

In order to complete the analysis, we are interested in the performance of PIM as a function of the values CH_i for $i \geq 1$ and CY_i for $i \geq 2$.

Lemma 4 *The cost of* PIM *on a connected graph of chords is at most*

$$\sum_{i=1}^{n}\frac{3i}{2}\cdot CH_i + \sum_{i=2}^{n}\left(\frac{3i}{2}-\frac{1}{2}\right)CY_i + \frac{1}{2}\ .$$

Proof. We assign costs to cycles and chains of OPT, such that the sum of all assigned costs is at least the total cost of PIM, possibly excluding a cost of $\frac{1}{2}$ incurred by PIM, which remains unassigned. Specifically, we assign a cost of $\frac{3}{2}$ to every edge, except every edge which has the property that it is the first edge ever removed of some cycle of OPT, and such an edge gets a cost of 1. Note that every cycle has at least one edge which is removed during the preprocessing, and hence a cycle of p edges has a total assigned cost of $\frac{3}{2}(p-1)+1 = \frac{3p}{2} - \frac{1}{2}$. Moreover, a chain of OPT with p edges, is assigned a cost of $\frac{3p}{2}$.

We analyze an algorithm which performs a single iteration of IM after the preprocessing phase is over. Clearly, similarly to the arc version, the algorithm may perform better by applying multiple iterations of IM. We next show that the entire cost of PIM is indeed assigned, except for possibly a residue of $\frac{1}{2}$ which remains unassigned.

Let d be the number of connected components resulting from removal of cycles. Let c be the number of such edges which are removed and have cost $\frac{3}{2}$. Let c' be the number of removed edges. The removal of the first edge of a cycle of OPT does not create a new connected component, therefore $d \leq 1 + c' - c$.

We next show that given a connected component of g edges (resulting from the removal of cycles), it is possible to partition it into a set of valid chains of length 2, and in addition, at most one chain of length 1. Therefore, an application of a single iteration of IM on this component would result in a cost of $3\frac{g}{2}$ if g is even and otherwise, the cost would be $3\frac{g-1}{2}+2 = \frac{3g+1}{2}$. Such a partition can be created inductively. Consider a DFS tree of a connected component, which contains at least two edges. If there exists a leaf which has at least two back edges, such a pair of edges can be chosen without violating connectivity, and can be routed to result in a valid chain. Otherwise, if some leaf has one back edge, pair this back edge with the edge to the parent. If no leaf has a back edge, take a leaf v of maximum depth, denote its parent by u. If u has at least one other child w, use the edges (u, v) and (u, w), which is possible since by definition, w is a leaf as well, and none of v and w has a back edge. If u has no other children, but there exists a back edge (u, x), use the pair of edges (u, x) and (u, v). Finally, if no such back edge (of u) exists, then u cannot be the root of the tree, and denote its parent by y. In this case, the pair of edges is (u, v) and (y, u).

We therefore get a total cost of $c' + 3\frac{|E|-c'}{2} + \frac{d}{2} \leq \frac{3|E|+1-c}{2}$. On the other hand, the total cost assigned to edges is $\frac{3|E|-c}{2}$, which proves our claim. ■

Theorem 5 *The approximation ratio of* COMBENUM *is at most* $\frac{7}{5} + \varepsilon$.

Proof. Note that $|E| = \sum_{i=1}^{n} iCH_i + \sum_{i=2}^{n} iCY_i > \frac{2}{5\varepsilon}$. Since $\text{OPT} = \sum_{i=1}^{n}(i+1)CH_i + \sum_{i=2}^{n} iCY_i$, we have

$$\begin{aligned}
COMB &= \min\{\text{PIM}, \text{DDAG}\} \leq \frac{1}{5} \cdot (4 \cdot \text{PIM} + \text{DDAG}) \\
&= \frac{1}{5} \cdot \left[\sum_{i=1}^{n} \left(4 \cdot \frac{3i}{2} + i + 3 \right) CH_i + \sum_{i=2}^{n} \left(4 \cdot \left(\frac{3i}{2} - \frac{1}{2} \right) + i + 2 \right) CY_i + 2 \right] \\
&\leq \frac{1}{5} \cdot \left[\sum_{i=1}^{n} (7i+7)\, CH_i + \sum_{i=2}^{n} 7iCY_i \right] + \varepsilon \left(\sum_{i=1}^{n}(i+1)CH_i + \sum_{i=2}^{n} iCY_i \right) = \left(\frac{7}{5} + \varepsilon \right) \cdot \text{OPT} \ .
\end{aligned}$$

■

4 Conclusion

In this paper, we addressed an open problem regarding the arcs version stated in [12], asking whether using both their methods, and methods of greedy preprocessing as in [4] could lead to

better results. We answer this question affirmatively. It is left as an open problem to find whether local-search can be helpful for the chord version.

References

[1] G. Călinescu, O. Frieder and P.-J. Wan, Minimizing electronic line terminals for automatic ring protection in general WDM optical networks, *IEEE Journal of Selected Area on Communications*, **20**, 183-189, 2002.

[2] G. Călinescu and P.-J. Wan, Traffic partition in WDM/SONET rings to minimize SONET ADMs, *Journal of Combinatorial Optimization*, **6**, 425-453, 2002.

[3] T. Eilam and S. Moran and S. Zaks, Lightpath arrangement in survivable rings to minimize the switching cost, *IEEE Journal of Selected Area on Communications*, **20**, 172-182, 2002.

[4] L. Epstein and A. Levin. Better bounds for minimizing SONET ADMs. *Journal of Computer and Systems Sciences*, **75(2)**, 122-136, 2009.

[5] L. Epstein and A. Levin. The chord version for SONET ADMs minimization. *Theoretical Computer Science*, **349(3)**, 337-346, 2005.

[6] M. Flammini, M. Shalom and S. Zaks, On minimizing the number of ADMs in a general topology optical network, *Discrete Applied Mathematics*, **157(12)**, 2701-2717, 2009.

[7] M. Flammini, M. Shalom and S. Zaks, On minimizing the number of ADMs - Tight bounds for an algorithm without preprocessing, *Journal of Parallel and Distributed Computing* **67(4)**, 448-455, 2007.

[8] O. Gerstel, P. Lin and G. Sasaki, Wavelength assignment in a WDM ring to minimize cost of embedded SONET rings, *Proc. INFOCOM 1998*, **1**, 94-101, 1998.

[9] C.A.J. Hurkens and A. Schrijver, On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems, *SIAM Journal of Discrete Mathematics*, **2(1)**, 68-72, 1989.

[10] L. Liu, X. Li, P.-J. Wan and O. Frieder, Wavelength assignment in WDM rings to minimize SONET ADMs, *Proc. INFOCOM 2000*, **2**, 1020-1025, 2000.

[11] A. Schrijver, *Combinatorial optimization polyhedra and efficiency*, Springer-Verlag, 2003.

[12] M. Shalom and S. Zaks, A $10/7 + \varepsilon$ approximation for minimizing the number of ADMs in SONET rings, *IEEE/ACM Transactions on Networking*, **15(6)**, 1593-1602, 2007.

[13] P.-J. Wan, G. Călinescu, L. Liu and O. Frieder, Grooming of arbitrary traffic in SONET/WDM BLSRs, *IEEE Journal on Selected Areas in Communications*, **18**, 1995-2003, 2000.