The master ring problem

Hadas Shachnai[1] and Lisa Zhang[2]

[1] *Computer Science Dept., Technion, Haifa 32000, Israel.*
[2] *Bell Labs, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974.*

We consider the *master ring problem (MRP)* which often arises in optical network design. Given a network which consists of a collection of interconnected rings $R_1, \ldots, R_K$, with $n_1, \ldots, n_K$ distinct nodes, respectively, we need to find an ordering of the nodes in the network that respects the ordering of every individual ring, if one exists. Our main result is an exact algorithm for MRP whose running time approaches $Q \cdot \prod_{k=1}^{K}(n_k/\sqrt{2})$ for some polynomial Q, as the n_k values become large. For the *ring clearance problem*, a special case of practical interest, our algorithm achieves this running time for rings of *any* size $n_k \geq 2$. This yields the first nontrivial improvement, by factor of $(2\sqrt{2})^K \approx (2.82)^K$, over the running time of the naive algorithm, which exhaustively enumerates all $\prod_{k=1}^{K}(2n_k)$ possible solutions.

Keywords: Master ring, shortest common supersequence, optical networks, exact algorithms.

1 Introduction

1.1 Problem Statement and Motivation

The prevalence of SONET (*Synchronous Optical NETwork*) technology has made the ring a popular network topology [13]. To carry a demand between two nodes on a SONET ring, traffic is routed simultaneously clockwise and counter-clockwise, one as the primary path and the other as the backup path. Often an optical network consists of a collection of interconnected SONET rings. A *master ring* contains every node in the network exactly once and respects the node ordering of every individual SONET ring. The *master ring problem (MRP)* is to find such a ring, whenever it exists.

Formally, the master ring problem is defined as follows. Suppose that a network consists of K rings, $R_1, \ldots, R_K$, with $n_1, \ldots, n_K$ distinct nodes, respectively. Each ring has two *orientations*, clockwise and counter-clockwise. We say that R is a *subring* of M (or M is a *master ring* of R) if either the clockwise or the counter-clockwise orientation of R can be obtained from M by erasing zero or more nodes from M. The goal is to find a master ring whenever it exists. Consider an instance of MRP as shown in Figure 1. The network consists of 3 rings. R_1 has the nodes $abcdef$, R_2 has the nodes $achg$, and R_3 has the nodes $ghcdi$. A possible master ring is $abghcdefi$.

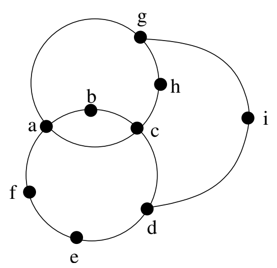

Fig. 1: An instance of MRP.

There are a number of reasons for finding master rings. For example, as a network evolves with growing traffic, it expands from an initially small number of SONET rings, to include a large collection of rings. Unfortunately, such expansion is often carried out in an ad-hoc manner, with circuits added and torn down over time. As a result, the network may have unnecessarily complex topology that makes network management a nightmare. To replace a spaghetti-like network, one simple topology is a master ring. Since a master ring respects the node ordering of every existing SONET ring, it has the advantage of preserving the routing label of every demand intra to an existing SONET ring. Indeed, a demand may traverse more nodes around the master ring than around its original SONET ring; however, preserving the order in which the SONET nodes are traversed allows to efficiently update the routing tables, rather than redefine from scratch the Label Switched paths. (Such paths are used, e.g., in MPLS [14].) Even if the network is not sought to be rebuilt, it still needs to handle the routine downtime, for purposes such as software upgrade. A master ring can then serve as a simple backup topology. Providing a master ring (whenever possible) to a network management system simplifies its operation and is therefore valuable [12, 1, 2].

We emphasize that the master ring can be viewed as a “logic” ring. That is, two neighboring nodes in the ring do not need to be physically connected by links already existing in the network. (Such links can be added once the master ring is set as a new/backup topology.) In addition, if two SONET rings R_i and R_j intersect then they have at least two nodes in common. This is because two common nodes can tolerate one node failure when supporting a demand between a node in R_i and a node in R_j.

One convenient way to represent the rings is to use sequences. Each orientation of a ring with n nodes corresponds to n *sequences*, depending on the node with which the sequence starts. Figure 2 shows the sequence representation of the instance in Figure 1. For example, the ring R_1 in Figure 2 has 6 clockwise sequences: $abcdef$, $bcdefa$, $cdefab$, $defabc$, $efabcd$, $fabcde$ and 6 counter-clockwise sequences: $fedcba$, $edcbaf$, $dcbafe$, $cbafed$, $bafedc$, $afedcb$. We also refer to each sequence as an *opening* of a ring. We say that S is a *subsequence* of T (or T is a *supersequence* of S) if S can be obtained from T by erasing zero or more symbols from T. Therefore, R is a *subring* of M (or M is a *master ring* of R) if some sequence that corresponds to R is a subsequence of a sequence that corresponds to M (see Figure 2).

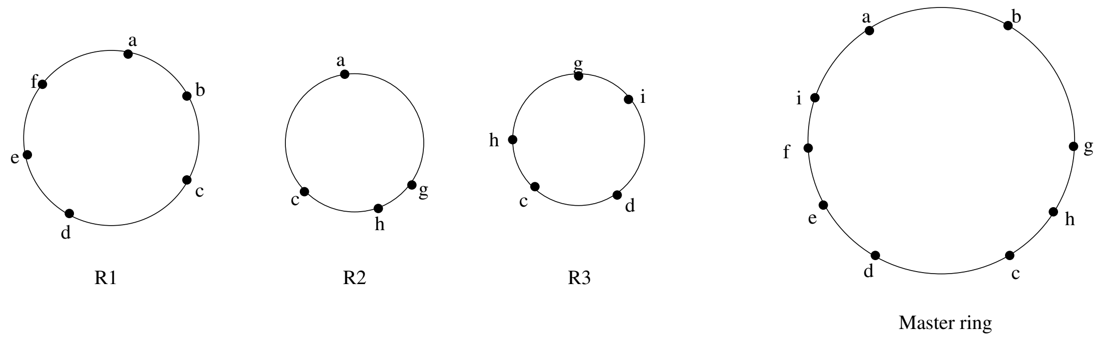

Fig. 2: (Left) Three rings R_1, R_2 and R_3. (Right) A possible master ring. For example, R_1 is a subring since its clockwise sequence $abcdef$ is a subsequence of the sequence $abghcdefi$ corresponding to the master ring; R_2 is a subring since its clockwise sequence $aghc$ is a subsequence; R_3's counter-clockwise sequence $ghcdi$ is a subsequence.

Given K sequences, each with any symbol appearing at most once, we note that it is easy to find a supersequence that contains each symbol once, if one exists. We construct a directed graph $G = (V, E)$, whose vertex set consists of the symbols in the K sequences and whose edge set consists of directed edges of the form (a, b), where a appears immediately before b in a sequence. Recall that a topological sort of a digraph is any linear order on the vertices respecting the graph's partial order. Hence, if G is acyclic then a topological sort of G is a (minimum possible length) supersequence of all K sequences. Deciding whether a digraph is acyclic and finding a topological sort are polynomially solvable (see e.g. [4]). Thus, our main task is to determine a set of sequences which can be represented as an acyclic digraph (whenever such a set exists). This is the focus of the paper.

1.2 Main Result

Our main result (in Section 2) is an exact algorithm for MRP, whose running time approaches $Q \cdot \prod_{k=1}^{K}(n_k/\sqrt{2})$ for some Q that is polynomial in the input size, as the n_k values become large. For the *ring clearance problem*, a special case of practical interest, our algorithm achieves this running time for rings of *any* size $n_k \geq 2$ (see in Section 3). This yields the first non-trivial improvement, by factor of $(2\sqrt{2})^K \approx 2.82^K$, over the naive algorithm which exhaustively enumerates all $\prod_{k=1}^{K}(2n_k)$ possible solutions (see, e.g., in [2]).

Our algorithm applies enumeration guided by an *intersection graph* of the network, which represents the interconnections among the rings. The graph is used for identifying subsets of rings whose openings leave only a few consistent openings for all other rings, thereby decreasing the remaining number of enumeration steps. While enumeration alone is inefficient, and using the intersection graph alone may result in a false solution for our problem (see in Section 2), we show that combining the two yields a significant improvement in running time, and guarantees that a master ring will be found, if one exists. We believe that similar techniques can be used in solving exactly other related problems, such as shortest common supersequence (SCS) and feedback arc set (FAS) and their variants. (See in Section 4.)

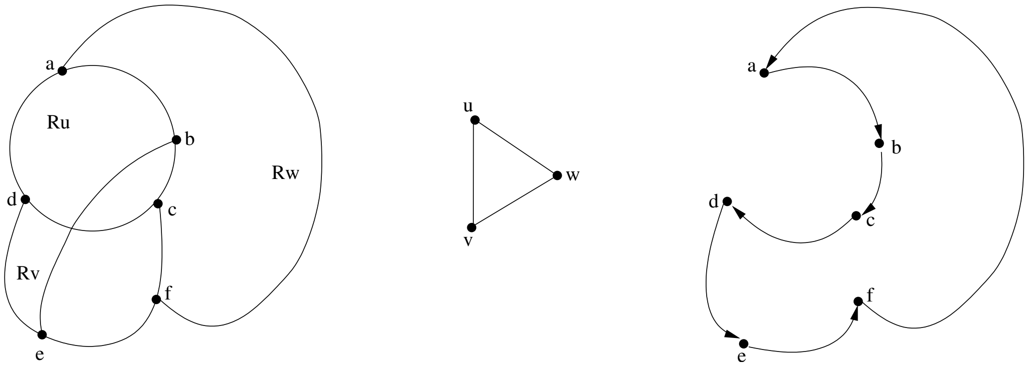

Fig. 3: (Left) An instance of MRP: R_u consists of nodes $abcd$, R_v consists of $cdef$ and R_w consists of $befa$. (Middle) The intersection graph H. (Right) R_u, R_v and R_w induce a large ring.

2 Algorithm

A naive solution for MRP is to enumerate all possible sequences for each ring and find if there is a topological sort for each resulting directed graph. Obviously, trying the total of $\prod_{1\le k\le K}(2n_k)$ possibilities suffices to solve the problem; the running time is $P \cdot \prod_{1\le k\le K}(2n_k)$, where P is the polynomial time required for topological sort. We describe below an algorithm which avoids enumerating some of these possibilities, by using the *intersection graph* of the network.

Before we apply our algorithm, we first eliminate all *singleton* nodes from each ring, i.e. those nodes that appear only in one ring. If node a is a singleton, then a can be ignored when constructing the master ring. Indeed, if a master ring exists without a, then a may always be added to the master ring. From now on we may assume without of loss of generality that every node appears in at least 2 rings.

We construct an undirected intersection graph H that shows how the rings are interconnected. The graph H consists of K vertices, each corresponding to one of the K rings. If two rings share common nodes, then there is an edge between their corresponding vertices in H. For clarity, we use *vertices* and *edges* when referring to the elements in the graph H and *nodes* and *links* – when referring to the elements of a ring. We also use letters near the beginning of the alphabet (such as a, b, c and d) when referring to nodes in a ring and letters near the end of the alphabet (such as u, v and w) when referring to vertices in H. For a vertex u in H, we use R_u to represent the corresponding ring. (See Figure 3 for an example.)

Our algorithm is motivated by observations that we detail later. Consider a vertex u in H. If R_v is already opened, and v is a neighbor of u, then the number of *consistent* openings of R_u is limited. (We say that a set of sequences are *consistent* if they have a supersequence.) For example, suppose that R_u and R_v have in common the nodes a and b, and R_v orders a before b; then, R_u would have to as well.

We note, however, that even if any two *neighboring* rings have consistent openings, it does not necessarily imply consistent openings for all rings. Consider the instance of Figure 3. When R_u is oriented clockwise, and R_v, R_w are oriented counter-clockwise, they induce a large ring $abcdef$. If R_u corresponds to the sequence $abcd$, R_v corresponds to $cdef$, and R_w corresponds to $efab$ then no opening of this induced ring contains the three sequences as subsequences. Therefore, these three openings cannot be consistent with one another. However, any two of these openings are consistent. (See Figure 3, Right). If, instead, R_w has the opening $abef$, then the three openings are consistent and have a master ring $abcdef$. The example in Figure 3 shows that we cannot use the graph H alone for determining good openings for all rings, since this graph indicates only the 'local' dependencies among the rings. To guarantee that no induced rings remain in the network after we open $R_1, \ldots, R_K$, we use the properties of the graph H only as guidance for the algorithm.

Algorithm $\mathcal{A}_{MR}$

0 Eliminate singleton nodes from $R_1, \ldots, R_K$. Construct the graph H with vertex set V.

Phase 1. Low-degree vertices

1 $N = L = \emptyset$.
2 While there is a low-degree vertex $v \in V - L - N$
  add vertex v to set L and its neighbors to set N.
3 For $u \in N$, try all possible sequences for R_u.
4 For $v \in L$ with x neighbors, try at most x possible sequences for R_v.
(See Lemma 1.)

Phase 2. Dominating set

5 Find a dominating vertex set D for the vertices $v \in H$ such that $v \in V - L - N$.
6 For $u \in D$, try all possible sequences for R_u.

Phase 3. Remaining vertices

7 Let $C = V - L - N - D$.
8 For $u \in C$, try a total of $y^{|C|}$ combinations of sequences for R_u,
where y is given in Lemma 5.
9 For each combination of sequences for vertices in $N \cup L \cup D \cup C$,
find a supersequence T using topological sort.
10 If T exists, a master ring is found. Algorithm terminates.

11 Output no master ring exists.

Fig. 4: The master ring algorithm $\mathcal{A}_{MR}$.

In our algorithm, $\mathcal{A}_{MR}$, we identify a low-degree vertex u in H and enumerate all possible openings of u's neighbors. Since u has low degree, relatively few rings are opened, but this dramatically limits the number of consistent openings of R_u. (See Lemma 1.) When H has only high-degree vertices, we find a *dominating set*, where a dominating set consists of vertices that are neighbors to every vertex not in the set. We can find a small dominating set in a graph with high degree vertices. By enumerating all possible openings for the (small number of) vertices in the dominating set, we can reduce the number of consistent openings for each remaining vertex by a constant factor. (See Lemma 5.) In our algorithm we define low and high degree vertices through a parameter δ; we set $\delta = \log n/c$, where $c \geq 3$ is some constant. If a vertex $u \in H$ has degree lower than δ then u is a *low-degree* vertex. A pseudocode of algorithm $\mathcal{A}_{MR}$ is given in Figure 4.

3 Analysis

For simplicity of exposition, we assume throughout the analysis that all of the rings are of the same size, n. Later, we show how the analysis extends to rings of arbitrary sizes.

In the following we show the correctness of algorithm $\mathcal{A}_{MR}$. Certainly, if the algorithm finds a sequence T that is a supersequence for some opening of every ring R_k, where $1 \leq k \leq K$, the master ring can be defined by T. However, since our algorithm does not exhaustively enumerate all of the $2n$ openings of each ring, if it does not find a supersequence we need to verify that we have not missed any opening that could have lead to a supersequence. We start by analyzing the first phase.

Lemma 1 *If a vertex $v \in L$ has x neighbors in H, and each neighbor is opened (i.e. is given a sequence), then at most x sequences of v can be consistent with the x neighboring sequences.*

Proof: Let u be a neighbor of v and S_u be the sequence representing the opening of the ring R_u. Consider the subsequence T_u of S_u that consists of the nodes common to R_u and R_v. Let a_u be the first symbol in T_u. Since we have no singleton nodes, we know that $\bigcup_u T_u$ contains all the nodes in R_v. Therefore, if S_v begins with a node in T_u for some neighbor u, then S_v has to begin with a_u; otherwise, S_v cannot be consistent with S_u. Furthermore, if S_v starts with a_u it has to follow the direction dictated by T_u. If T_u

consists of 3 or more nodes, then this direction is unique. If T_u consists of 2 nodes, then either clockwise or counter-clockwise direction could be consistent. We examine the two neighboring nodes b and c of a_u on ring R_u that are not in T_u. For S_v to start at a_u and continue with b, b has to be the first node in some other subsequence $T_{u'}$ for some neighbor u', or S_v cannot be consistent with $S_{u'}$. In addition, if both b and c are the first nodes of some subsequences, no matter which direction S_v takes, S_v cannot be consistent with both. Therefore, S_v can only start with one of at most x nodes and for each starting node there is only one possible direction. □

From Lemma 1, in Line 4 of the algorithm we try at most δ sequences for any ring R_v, such that v is a low-degree vertex in H. This allows us to bound the running time of Phase 1.

Lemma 2 *The running time of Phase 1 is at most* $(1 + o(1))\dfrac{n^{\hat{k}}}{2^{\hat{k}(c-2)}}$ *where* $\hat{k} = |L| + |N|$ *is the total number of vertices dealt with in this phase.*

Proof: It is easy to see that the number of combinations that Phase 1 tries is bounded by $(2n)^{|N|}x^{|L|}$, where $x \leq \delta$. However, to execute Line 4 we need to determine the orientation for each of the x potential openings of a ring. This can be done in time $O(\alpha x|L|)$ for some constant α using the procedure described in Lemma 1. Therefore, the outer loop in our algorithm, Phase 1, takes at most $(2n)^{|N|}(\alpha x|L| + x^{|L|})$, which is $(1 + o(1))(2n)^{|N|}\delta^{|L|}$. Note that the $o(1)$ term is a function of the ring size n and c,

Let us look at the running time more closely. Let $\hat{k} = |L|+|N|$ be the total number of vertices handled in Phase 1. Since L consists of vertices with degree lower than δ, we have $|L| \geq \hat{k}/\delta$ and $|N| \leq \hat{k}(1 - 1/\delta)$. Therefore,

$$(2n)^{|N|}\delta^{|L|} \quad = \quad n^{|L|+|N|}\left(\frac{\delta}{n}\right)^{|L|} 2^{|N|} \leq n^{\hat{k}}\left(\frac{\delta}{n}\right)^{\hat{k}/\delta} 2^{\hat{k}(1-\frac{1}{\delta})} = n^{\hat{k}} 2^{\hat{k}(\frac{log\delta}{\delta}-c)} 2^{\hat{k}(1-\frac{1}{\delta})} \leq \frac{n^{\hat{k}}}{2^{\hat{k}(c-2)}}.$$

□

Let us bound the size of the dominating set D in Phase 2.

Lemma 3 $|D| \leq |V - L - N| \cdot \frac{1+\ln(1+\delta)}{1+\delta}$.

Proof: We first prove the next claim, which generalizes a result in [3]. Let $G = (V, E)$ be a graph, such that all the vertices in $V' \subseteq V$ have degree at least s. Then there exists a subset of vertices $V'' \subseteq V'$ of size at most $|V'|\frac{1+\ln(1+s)}{1+s}$, such that $U = (V \setminus V') \bigcup V''$ is a dominating set for G.

Consider the following Greedy algorithm. (i) We start by adding all the vertices in $V \setminus V'$ to U. (ii) Let W be the set of vertices in V' that are not in U and do not have a neighbor in U; While $|W| > |V'|/(s+1)$ do: Find a vertex $v \in W$ such that v has a maximal number of neighbors in W; add v to U. (iii) Add the vertices in W to U.

Clearly, U is a dominating set for G. To bound the size of V'', we first note that, by an averaging argument, since all the vertices in $V \setminus V'$ are added to U, the number of iterations until $|W| \leq |V'|/(s+1)$ is at most $|V'| \ln(s+1)/(s+1)$. (A similar argument is given in the analysis of the deterministic algorithm for the dominating set problem in [3]; we omit the details.) Hence, we get that $|V''| \leq |V'| \ln(s+1)/(s+1) + |V'|/(s+1)$. If we set $V' = V - N - L$ and $s = \delta$, our lemma follows directly. □

The greedy algorithm in Lemma 3 takes time at most quadratic in K, the number of vertices in H. Hence,

Lemma 4 *The running time of Phase 2 is* $poly(K) + (2n)^{|D|}$.

We now discuss how to efficiently find openings for the remaining vertices in C during Phase 3.

Lemma 5 *The running time of Phase 3 is at most* $poly(K)y^{|C|}$*, where* $y \leq \sqrt{2}(3 + n/2)$*. Hence, the running time of phase 3 is* $poly(K)(n/\sqrt{2})^{|C|}$.

Proof: During Phase 3, every vertex $u \in C$ has some neighbor v in the dominating set D. By assumption, R_u and R_v have at least 2 nodes, say a and b, in common. Any sequence of R_v defines an ordering of a and b, i.e. a appears before b or after b. Among the $2n$ sequences of R_u, exactly n respect this ordering of a and b. Any of the other n sequences that disrespect the ordering cannot produce a topological sort and therefore need not be considered. We get that it suffices to enumerate at most n sequences for the ring R_u, for any $u \in C$.

We can further reduce the number of enumerations using the *pairing algorithm* described below. Instead of directly enumerating n possible sequences for R_u where $u \in C$, we pair up the sequences so that one sequence in a pair begins with a node, say a, and the other sequence in the pair ends with the node a. We refer to a as the *pivot* of the pair. (At most 3 out of the n sequences cannot be paired up with another sequence.) More concretely, let us consider the following example. For $u \in C$, suppose the ring R_u has 6 nodes $abcdef$ clockwise. Suppose that u's neighbor $v \in D$ has chosen a sequence for R_v in which b precedes e. Therefore, any sequence for R_u needs to have b before e, else there is no topological sort. Among the 12 possible sequences for R_u, the following 6 have b before e.

$$abcdef \qquad bcdefa \qquad fabcde \qquad dcbafe \qquad cbafed \qquad bafedc$$

We pair up the 1st and 2nd sequences, $abcdef$ and $bcdefa$, with pivot a, and pair up the 4th and 5th sequences, $dcbafe$ and $cbafed$ with pivot d. The 3rd sequence, $fabcde$, and the 6th sequence, $bafedc$, remain singletons.

We first enumerate the $3 + n/2$ groups (at most $n/2$ pairs and at most 3 unpaired singletons) for every $u \in C$. This gives a total of at most $(3+n/2)^{|C|}$ possibilities. In the following we show that to determine the actual sequence within each pair for any $u \in C$, we do not need to try both possibilities. In fact, a total of $2^{|C|/2}$ trials suffices. Hence, the total number of trials is $(3 + n/2)^{|C|}2^{|C|/2}$, which implies $y \leq \sqrt{2}(3 + n/2)$.

For example, suppose ring R_u where $u \in C$ has the above 6 possibilities and we are considering the first pair with pivot a. Since a is not a singleton node, a necessarily appears in another ring, say R_w. If the sequence for R_w is decided, then we necessarily know which sequence, $abcdef$ or $bcdefa$, would be good. This is because R_u and R_w must share a node other than a and let's call this node c. Since a is a pivot for R_u, if a appears before c for R_w then only the first sequence $abcdef$ can be good; if a appears after c for R_w then only the second sequence $bcdefa$ can be good.

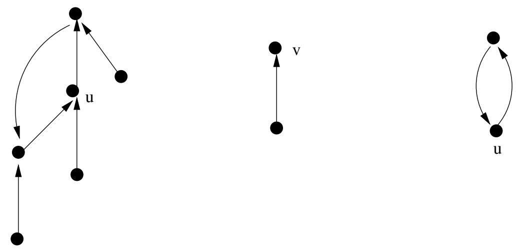

Fig. 5: Examples of the graph F. One possible solution for the graph on the left (right) is to circle the vertex u and mark all other vertices cross. If vertex v in the middle graph is not in C then neither vertex is circled.

In general, we construct a directed graph F where each vertex corresponds to a vertex in C. We put a directed edge from u to w if the pivot of u is a vertex in the ring R_w. If there are multiple such rings R_w for u we choose an arbitrary one. As argued above, if there is a directed edge from u to w, then we only need to enumerate the two choices in a chosen pair for R_w and and the choice for R_u is implied. We determine which rings to enumerate as follows. We mark a *cross* on a vertex to indicate that the choice is implied and we mark a *circle* on a vertex to indicate that we enumerate both possibilities. Initially, we mark a cross on a vertex u if it has no outgoing edges. This means the pivot of u appears in some ring R_w that belongs to $L \cup N \cup D$. Hence, the sequence for R_w is already chosen and therefore the sequence for R_u is implied. For each vertex u in F that is not yet marked, we follow the directed edges, starting from u, until (i) we have reached a marked vertex (either with a circle or with a cross), or (ii) we stop right before the path from u intersects itself, i.e., in a vertex z such that there is an edge (z, u). In the latter case, we circle the vertex where we stop. In both cases, we also mark a cross on every (unmarked) vertex along the path. (See Figure 5.)

It is easy to verify that the choice for each vertex with a cross can be implied from the choice for some vertex with a circle. In terms of the running time, we observe that at most half of the vertices in F can be circled, since each circled vertex needs at least one distinct vertex that has a cross. To mark each vertex in F with a circle or cross requires visiting each vertex once. Hence, the time requirement is linear in $|C|$. It follows that the running time of Phase 3 is at most $poly(|C|)(3 + n/2)^{|C|} \cdot 2^{|C|/2}$, which is $poly(K)(n/\sqrt{2}))^{|C|}$. □

From Lemmas 1 and 5 we see that although algorithm $\mathcal{A}_{MR}$ does not enumerate all possibilities for the vertices in L and C we do not miss out any potentially good opening. Our algorithm is therefore correct. We bound the running time as follows.

Theorem 6 *When the ring sizes n gets large, the running time of our algorithm $\mathcal{A}_{MR}$ is $(1+o(1))(n/\sqrt{2})^K \cdot P \cdot Q$, where P is the time needed for topological sort of K sequences of length n, and Q is a polynomial in K.*

Proof: It is easy to see that the overall running time is the product of the running times of the three phases and P, the time for each topological sort. From Lemmas 2, 4 and 5, the overall running time is,

$$(1+o(1))\frac{n^{|L|+|N|}}{2^{(|L|+|N|)(c-2)}} \cdot (poly(K) + (2n)^{|D|}) \cdot (poly(K)(n/\sqrt{2})^{|C|}) \cdot P.$$

We have,

$$\frac{n^{|L|+|N|}}{2^{(|L|+|N|)(c-2)}} \cdot (2n)^{|D|} \cdot (n/\sqrt{2})^{|C|} \quad \leq \quad \frac{n^K}{2^{K/2}} \cdot \frac{1}{2^{(|L|+|N|)(c-2.5)}2^{-3|D|/2}}.$$

When the ring size n gets large, the value of δ is large and hence the size of the dominating set D approaches a small constant. When $c > 2.5$, the exponent of the second term in the above denominator is positive. Hence, the above expression approaches $(n/\sqrt{2})^K$. Therefore the overall running time of algorithm $\mathcal{A}_{MR}$ is $(1+o(1))(n/\sqrt{2})^K \cdot P \cdot Q$. □

We note that the naive algorithm that enumerates all $2n$ possibilities for each ring takes $(2n)^K \cdot P$ time. Our algorithm essentially improves the term $(2n)^K$ to $(n/\sqrt{2})^K$.

Our algorithm achieves better running time for two important subclasses of inputs. Consider the subclass of *sparse* inputs: in the intersection graph of the rings, H, all the vertices are of low-degree. Thus, our algorithm terminates after Phase 1. The following comes directly from Lemma 2.

Corollary 7 *For any $c \geq 3$, if the maximal degree in H is smaller than $\log n/c$ then the running time of the algorithm is at most $(1+o(1))(\frac{n}{2^{c-2}})^K$. In particular, if the maximal degree in H is some constant $d \geq 1$ then the running time of $\mathcal{A}_{MR}$ is $(1+o(1))(n^{1-\frac{1}{d}})^K$.*

Consider now the subclass of *dense* inputs, where each node in the network appears in at least m rings, for some $m \geq 2$; then, in Phase 3 of our algorithm, we get that the remaining rings can be grouped to 'clusters' of size at least m. In each cluster we need to try the two possible openings of a single ring. (We use as before the algorithm of Phase 3, with slight modifications. We give the details in the full version of the paper.) This reduces the running time of Phase 3 to $poly(K)(3+\frac{n}{2})^{|C|} \cdot 2^{\frac{|C|}{m}}$.

Corollary 8 *If each node in the network appears in at least m rings, for some $m \geq 2$, then the running time of the algorithm is at most $(1+o(1))(\frac{n}{2^{(1-\frac{1}{m})}})^K$.*

Ring Clearance. In the ring clearance problem, we need to "clear" R_1 and reroute all the traffic through the other rings. In order for such transition to occur, it is assumed that R_1 intersects with each of the other rings. In other words in the intersection graph H every vertex is a neighbor of the vertex w corresponding to R_1. Hence, $\{w\}$ is a dominating set for all vertices in H. We only need to apply Phase 3 of our algorithm. Using the simple analysis in Lemma 3, it is easy to see that any opening of R_1 limits the number of openings of any other ring to at most n. If we follow the more sophisticated pairing argument in Lemma 3 we only need to try a total of $(n/\sqrt{2})^K$ possibilities.

Corollary 9 *The algorithm solves the ring clearance problem in at most $(n/\sqrt{2})^K \cdot P \cdot Q$ steps, for rings of any length $n \geq 2$.*

Rings of distinct lengths. The analysis for the case where each ring R_u has a distinct size, n_u, is similar. We remove the singleton nodes and create the intersection graph H as before. For Phase 1, we say that a vertex u has low degree if it has fewer than $\delta_u = \log n_u/c$ neighbors. The running time of Phase 1 is at most $(1+o(1))\prod_{u\in N}(2n_u)\prod_{u\in L}\delta_u$. Similar to Lemma 2 we deduce,

$$\prod_{u\in N}(2n_u)\prod_{u\in L}\delta_u \leq \frac{\prod_{u\in L\cup N} n_u}{2^{(|L|+|N|)(c-2)}}.$$

In Phase 2, we find again a dominating set D and we can bound $|D|$ by $|V-L-N| \cdot \frac{1+\ln(1+\delta)}{1+\delta}$, where $\delta = \min_u \delta_u$. When all ring sizes n_u get large, the size of D approaches a small constant. The running time for Phase 2 is $poly(K) + \prod_{u \in D}(2n_u)$. Finally, for Phase 3 we use the pairing algorithm as described in Lemma 5 for the vertices in C, and the running time is $poly(K) \prod_{u \in C}(n_u/\sqrt{2})$.

Theorem 10 *For rings with distinct sizes, when the ring sizes get large the running time of algorithm $\mathcal{A}_{MR}$ is $(1 + o(1)) \prod_u (n_u/\sqrt{2}) \cdot P \cdot Q$, where the $o(1)$ term is a function of the ring sizes and c, P is the time needed for topological sort of K sequences, and Q is a polynomial in K.*

4 Relation to Other Problems

We briefly discuss how MRP relates to the shortest common supersequence (SCS) and feedback arc set (FAS) problems. We defer the details of this section to the full version.

4.1 Shortest Common Supersequence

In SCS we are given K strings, $S = \{S_1, \ldots, S_K\}$, of lengths $n_1, \ldots, n_K$, over an alphabet Σ, where $|\Sigma| = \mathcal{N}$. We seek a supersequence T for S of minimum length. MRP defines the following natural variant of SCS. A *two-way cyclic* permutation of a string allows cyclic shifts of the string in the forward and reverse directions. For example, the string $abcd$ has 4 forward shifts, $abcd$, $bcda$, $cdab$ and $dabc$, and 4 reverse shifts, $dcba$, $cbad$, $badc$ and $adcb$. In the *two-way cyclic SCS (2Cyclic-SCS)* problem, we seek a string T of minimum length, such that there exists a two-way cyclic permutation of each string $S_1, \ldots, S_K$ in S that is a subsequence of T. We say that T is a *2cyclic supersequence* for S. A supersequence T of length $\mathcal{N}$ corresponds to a master ring for the set of rings defined by $S_1, \ldots, S_K$.

The SCS problem is known to be hard to approximate. In particular, Jiang and Li [10] showed that there exists a constant $\varepsilon > 0$ such that if SCS has a polynomial time approximation algorithm with ratio $\log^{\varepsilon} K$, then NP is contained in DTIME($2^{polylog(K)}$). The best known approximation ratio is $\frac{K+3}{4}$, due to Fraser and Irving [7]. Middendorf considered in [11] a number of variants of SCS. This includes the Cyclic-SCS problem, in which the strings in S can be cyclically permuted in the *same* direction. The paper shows that this problem is NP-hard. (Cyclic-SCS solves MRP in the case where each ring has a *fixed* orientation.) On the other hand, Permutation-SCS, where each string S_k can be permuted to any one of the $n_k!$ possibilities, is shown in [11] to be polynomially solvable for strings of any length. This implies that MRP can be solved in polynomial time for inputs where $n_k \leq 3$, for $1 \leq k \leq K$.

The hardness of 2Cyclic-SCS can be shown via a reduction from the vertex cover problem.

Theorem 11 *Given m, it is NP-hard to determine if 2Cyclic-SCS has a solution at most m.*

Our algorithm, $\mathcal{A}_{MR}$, can be combined with a dynamic programming algorithm for SCS [6, 9, 5] to yield an optimal solution for 2Cyclic-SCS with a running time of $O(\mathcal{N} 2^K \prod_{k=1}^{K} n_k^2)$. Alternatively, we can find a supersequence T of minimum length by *guessing* first the cyclic shift of each string in T; we can then solve the SCS problem using dynamic programming (see, e.g. [6]). The best known DP algorithm has running time $O(\prod_{k=1}^{K} n_k)$. Thus, we have,

Theorem 12 *The 2Cyclic-SCS problem can be solved in $O(\frac{\prod_{k=1}^{K} n_k^2}{2^{K/2}})$ steps.*

4.2 Feedback Arc Set

MRP relates also to the *feedback arc set (FAS)* problem in directed graphs, which is known to be NP-hard [8]. Consider the special case of MRP in which the orientation for each of the rings is given. We denote this *oriented* version $MRP_{\mathcal{O}}$. We can view $MRP_{\mathcal{O}}$ as the following variant of FAS, that we call *exact subset FAS*. We have a directed graph $G = (V, E)$, and a set of K (directed) cycles in G, $R = \{R_1, \ldots, R_K\}$. Let $G' = (V', E')$ be the subgraph induced by the vertices and edges in R. We seek a subset of K edges in E' whose deletion leaves G' acyclic, such that in each of the cycles $R_1, \ldots, R_K$ we omit *exactly* one edge. Such a subset of vertices exists iff we have a solution for the corresponding $MRP_{\mathcal{O}}$ instance. Since we are given the orientation for each of the rings, we can apply only Phase 3 of algorithm $\mathcal{A}_{MR}$. By finding a master ring, we solve the *exact subset FAS* problem. Hence, we have

Corollary 13 *For any $K \geq 1$ and $n_k \geq 2$, for all $1 \leq k \leq K$, exact subset FAS on the subgraph G' induced by K cycles of the lengths $n_1, \ldots, n_K$ can be solved in $P \cdot (\prod_{k=1}^{K} n_k/(\sqrt{2})^K)$ steps, where P is a polynomial of K.*

5 Open Problems

Consider the following parameterized version of the Permutation-SCS. Each string S_k, $1 \leq k \leq K$, is associated with a subset of permutations, Π_k, and we seek a supersequence T of minimum length, such that there exists a permutation of S_k in Π_k that is a subsequence of T. We call this problem *Perm-SCS*(Π_k). Indeed, the Cyclic-SCS problem is a special case of this problem, in which Π_k is the set of n_k cyclic shifts of S_k, in a single direction. As shown in [11], this special case of the problem is NP-hard. We have shown (in Theorem 11) that if we extend the permutation sets in the Cyclic-SCS, so that Π_k is the set of cyclic shifts in *two* directions (2Cyclic-SCS), the problem remains NP-hard. On the other hand, when Π_k is the set of *all* possible permutations of S_k (Permutation-SCS), the problem is solvable in polynomial time [11]. Determining whether Perm-SCS(Π_k) is polynomially solvable on other classes of inputs remains an open problem.

Finally, a natural variant of MRP which is of practical interest, is to identify a *maximum* subset of rings for which we can find a master ring, in any given network.

Acknowledgements

We thank an anonymous referee for helpful comments on the paper.

References

[1] S. Acharya, B. Gupta, P. Risbood, A. Srivastava. *Hitless Network Engineering of SONET Rings*, Globecom 2003.

[2] S. Acharya, B. Gupta, P. Risbood, A. Srivastava. *In-service Optimization of stacked SONET Rings*, submitted.

[3] N. Alon and J. H. Spencer. The Probabilistic Method, Second Edition. Wiley-Interscience, 2000.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, 2nd Edition, MIT Press and McGraw-Hill, 2002.

[5] D. E. Foulser, M. Li and Q. Yang, *A Theory of Plan Merging*, Artificial Intelligence, 57, 1992, pp. 143–181.

[6] C. B. Fraser, *subsequences and Supersequences of Strings*. Ph.D. Thesis, Dept. of Computer Science, University of Glasgow, 1995.

[7] C. B. Fraser and R. W. Irving , *Approximation algorithms for the shortest common supersequence*, Nordic J. Comp. 2, 1995, pp. 303–325.

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

[9] S.Y. Itoga, *The String Merging Problem*, BIT, 21, 1981, pp.20–30.

[10] T. Jiang and M. Li, *On the Approximation of Shortest Common Supersequences and Longest Common Subsequences*, SIAM Journal on Computing, 24(5), October 1995, pp. 1122–1139.

[11] M. Middendorf, More on the complexity of common superstring and supersequence problems, *Theoretical Computer Science* 125 (1994), 205-228.

[12] Mobius network management and optimization systems. Lucent Technologies Proprietary. Internal website: http://www-zoo.research.bell-labs.com/∼mobius/.

[13] R. Ramaswami and K. Sivarajan. *Optical networks: a practical perspective*. (Morgan Kaufmann Publishers Inc., San Francisco, 1998).

[14] E. Rosen and A. Viswanathan *Internet Standards for Multi Protocol Label Switching*. In http://www.ietf.org/rfc/rfc3031.txt