Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Heurisztikus algoritmus tervezése elosztott ponthiba lokalizálásra

A heuristic algorithm for network-wide local unambiguous node failure localization

TDK DOLGOZAT

Készítette
Gyimóthi László

Konzulens
dr. Tapolcai János

October 22, 2014

Contents

Abstract in Hungarian

Napjainkra az Internet megbízhatósága kulcskérdéssé vált. A hatalmas sávszélesség igényt kielégíteni képes optikai gerinchálózatok állandó felügyelete, valamint az esetleges hibák gyors detektálása, lokalizálása és javítása alapvető követelmény a szolgáltatók számára. Mivel a hibainformáció terjesztésre szolgáló elektromos jelzésrendszeri üzenettovábbítás a helyreállítási idővel szemben támasztott követelményekhez képest jelentős, felmerül az optikai réteg hibalokalizációs képességének szükségessége. Az optikai hibalokalizáció egy lehetséges megvalósítása ún. monitorozó utak használata, amelyek mindössze 1-1 bit információt hordozva kódolják, azonosítják a hálózat elemeit. A monitorozó utak segítségével a hálózatban megvalósítható az elosztott ponthibalokalizáció, vagyis a hálózatnak azon képessége, hogy bármely csomópont bármely másik csomópont hibáját detektálni, és egyértelműen lokalizálni tudja, ezáltal rendkívül gyors hálózati helyreállítás garantálható.

Dolgozatomban egy általam megalkotott heurisztikus algoritmust mutatok be, melynek segítségével elosztott ponthiba-lokalizáció hajtható végre optikai gerinchálózatokban. A javasolt rekurzív algoritmust valós hálózati topológiákon, illetve általam generált síkgráfokon egyaránt teszteltem, és a szakirodalomban megtalálható eddig ismert eredményekhez képest jelentős javulást sikerült elérni mind futási időben, mind célfüggvény értékben.

Abstract

Reliability of the Internet became crucial in the past years. The permanent monitoring, surveillance of the optical backbone networks along with failure detection, localization and correction functions are fundamental requirements, that an operator must satisfy. The traditional approach for failure detection and alarm message dissemination uses control plane packets, that generates a significant delay. A possible solution for implementing optical layer failure localization in the network is using multihop supervisory lightpaths, called monitoring trails. These trails intrinsically carry a single bit of information, representing the statuses of the network elements they traverse by. By properly defining and utilizing monitoring trails, all nodes can unambiguously identify the faulty element in the network, almost instantaneously.

In this study I introduce a heuristic algorithm that achieves network wide unambiguous failure localization in optical networks. The proposed recursive algorithm had been validated on a large number of topologies. Results show, that it significantly outperforms the previously known algorithms for the given problem.

Introduction

The problem investigated in this study is related to optical layer restoration, which allows full reconfiguration of the optical network to survive failures. Restoration is considered a rather theoretical approach which requires only the minimal protection capacity. Implementing restoration requires an intensive synchronization in the restoration process within 50msec after the failure. For example implementing stub release[1] in real time requires an exact knowledge of the failed elements in the network for each node. Fast failure localization is currently not feasible and the main target of this work.

The traditional approaches use control plane packets to exchange the status information after failures. A very general approach is to fix the set of failure states the network can survive at planning. These failure states are often called *shared risk link groups* (SRLG), and typically are single link or node failures. The rest of failure events are unlikely to happen and thus ignored at the planning stage. As a result, the failure state is a little information that should be disseminated in a very short time. An alternative idea was to follow the compressed sensing approach, where a set of supervisory lightpaths are allocated in the network. Each supervisory lightpath, often called *monitoring-trails* (m-trails), intrinsically carries one bit of information about the failure. The idea is that a network failure interrupts every supervisory lightpaths traversed by. This is immediately seen at every downstream node to the failure along the interrupted supervisory lightpaths with optical channel tap monitors. A node is called local unambiguous failure localization (L-UFL) capable if this information is sufficient to identify the failed SRLG. The m-trail approach is expected to serve as a complement to the existing electronic signaling approaches and enables an ultra-fast and deterministic fault management process [2,3,7,9–12,14].

In this study we follow the model of [11], where every node should be L-UFL capable, so the network can perform network-wide L-UFL (NL-UFL) [10], and the target is to localize single node failures. The main contribution of the study is to present a new heuristic algorithm for the problem faster then the prior art.

The present study is organized as follows. Chapter 1 presents a brief overview on optical layer failure localization techniques and defines the m-trail problem. It also contains a short literature review and presents the background knowledge for the research. Chapter 2 introduces a novel heuristic, called RMCA, that solves the NL-UFL problem for single node failures on general graphs. Chapter 3 shows simulation results on a wide range of topologies, which verify the proposed algorithm. Chapter 4 concludes the work.

[1] releasing the bandwidth of all the connections interrupted by the failure

Chapter 1

Background

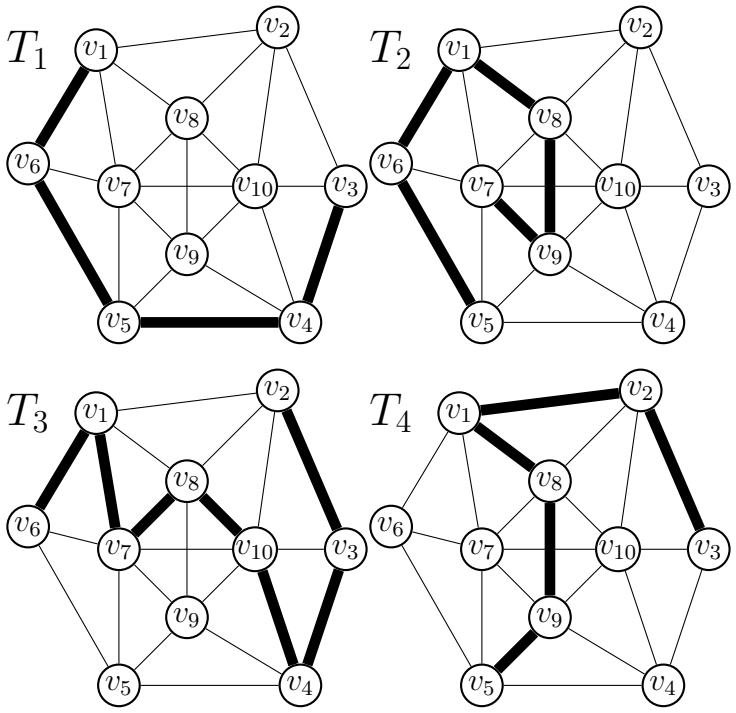
The issue of fast failure localization in optical backbone networks is a crucial problem for network operators. Establishing failure recovery in the optical layer isolated from the upper layer recovery protocols is a fundamental requirement in order to avoid the initiation of multiple concurrent mechanisms. A frequently used scheme is to settle monitors in the network, that can observe the statuses of certain network entities. In general, these monitors are deployed at the network nodes, and their task is to generate a control plane alarm message whenever an irregularity in the network is detected. The dissemination of these alarm messages provides failure localization for all the nodes in the network. For modeling possible network failures, often the concept of SRLG (Shared Risk Link Group) is used: an SRLG contains a set of network entities (nodes, links, software entities, etc.), that share a common risk of single failure, meaning that a failure in the network is likely to affect the elements in the same SRLG. In this study we focus on single node failures, i.e. each SRLG contains solely one node.

We assume, that a node's failure interrupts all of its interfaces, so that the faulty node becomes isolated from the network. Hence, the 2-connectivity of the networks can be assumed, as if this assumption is not satisfied, the failure of a cut vertex would separate the network into two, or more components, making the failure restoration process impossible.

1.1 Centralized Unambiguous Node Failure Localization (UNFL)

Network monitors can observe on-off statuses of dedicated multihop supervisory lightpaths, called *monitoring trails* (m-trails). These m-trails traverse a set of nodes and links in the network. If we model the network as an undirected graph $G = (V, E)$, then an m-trail T_1 can be represented as a connected subgraph of G. There are two distinct scenarios in terms of monitoring trail usage for failure localization. In the *centralized* scheme, certain monitoring nodes on each trail generate control plane messages whenever a failure occurs on the given m-trail. The alarm messages are collected by a central controller entity, that can unambiguously localize the faulty node from the given monitoring trail statuses (as mentioned before, we consider only single node failures in this study). As an m-trail intrinsically carries only a single bit of information (i.e. its on-off status), in order to un-

Figure 1.1: *ACT of an Unambiguous node failure localization (UNFL)*



Failure	T_1	T_2	T_3	T_4
v_1	1	1	1	1
v_2	0	0	1	1
v_3	1	0	1	1
v_4	1	0	1	0
v_5	1	1	0	1
v_6	1	1	1	0
v_7	0	1	1	0
v_8	0	1	1	1
v_9	0	1	0	1
v_{10}	0	0	1	0

ambiguously identify any failure of the network, a set of m-trails $\mathcal{T} = \{T_1, \ldots, T_b\}$ has to be established. Once the exact corresponding subgraphs of the m-trails are defined, each vertex can be assigned a binary code. The i-th position of the binary code of node v is 1 iff T_i traverses v. As our aim is to implement Unambiguous Node Failure Localization (UNFL), a unique binary code has to be assigned to all the nodes. The codes' uniqueness guarantee that all nodes are traversed by a different set of trails, thus the centralized controller can unequivocally identify the faulty node. If we denote the number of nodes with n, and the number of m-trails with b, then the arrangement of these binary codes results in a matrix with n rows and b columns, called an Alarm Code Table (ACT). Figure 1.1 shows an example of UNFL in a 10-node network. Edges denoted with thick lines represent the monitoring trails. One might observe, that in this particular case each m-trail is a *path*, meaning it is a sequence of vertices and edges. This is not an obligatory property of the monitoring trails, in our study the only restriction on the defined lightpaths is their connectivity. Note, that all nodes are assigned a unique binary string, thus any single node failure can be unambiguously identified by the controller.

1.2 Network-wide Local UNFL

The centralized scenario creates a large signaling overhead, as all the generated alarm messages have to be collected by the controller. The whole process could be sped up, if there are only a few nodes that have to change status information via alarm messages. Ideally, there is a node, that can inspect all the trails, thus could operate as a controller without collecting any alarm messages.

We define *local information* for a node v as the on-off statuses of all those lightpaths that v is traversed by. From a practical point of view, every node can obtain these statuses of their traversing lightpaths via optical signal tapping. If there is a node v, that can perform UNFL using only its local information, then v is said to be Local-UNFL (L-UNFL) capable. For instance, node v_1 on Figure 1.1 is L-UNFL capable, as all the four trails are traversing it. If all the nodes in the network are L-UNFL capable, we say that Network-wide L-UNFL (NL-UNFL) is guaranteed.

The length of a monitoring trail T_i is denoted as $|T_i|$, and can be measured as the number of nodes it traverses. Our main objective is to implement NL-UNFL for general topologies, with minimum total cover length, that is $\|T\| = \sum_{i=1}^{b} |T_i|$. The cover length is also called bandwidth cost, as this implies the additional bandwidth consumption of monitoring.

The simplest way to implement NL-UNFL in a 2-connected network with n nodes is to define n monitoring trails where the i-th trail contains all the nodes except for the i-th one. Thus, when a single failure occurs in the network, it affects all the trails except for the one that bypasses the faulty element, so that every node can unambiguously identify the location of the failure (see Figure 1.2). This approach provides a fast and clear way to reach the desired operation, however, as the number of required m-trails is a linear function of the size of the graph, and the normalized cover length is $n - 1$, this configuration does not satisfy our requirements and constraints on the bandwidth occupancy. Instead of linear results, we seek after solutions with logarithmic nature, as only $\lceil \log_2 n \rceil$ bits are needed to distinguish n elements.

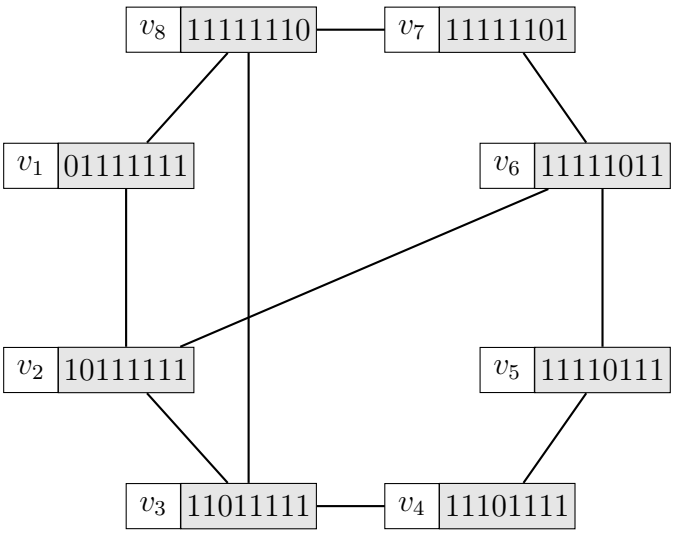


Figure 1.2: *Implementing NL-UNFL in a simple way on a 8-node random 2-connected graph*

1.3 Problem Definition

The problem input is an undirected graph $G = (V, E)$ with node set V and link set E, where the number of nodes is denoted by $n = |V|$. The NL-UFL m-trail allocation problem for single-node failure (NL-UNFL) is to establish a set of m-trails, denoted by

Table 1.1: *Notation list*

Notation	Description
$G = (V, E)$	undirected graph representation of the topology
$n = \|V\|$	the number of nodes in G
b	the number of m-trails
$\mathcal{T} = \{T_1, \ldots, T_b\}$	a solution with b m-trails
$\mathcal{T}^j$	the m-trails seen at node v_j
T_i	the i^{th} m-trail, which is a set of nodes in G
$\|T_i\|$	number of nodes the i^{th} m-trail traverses
b_j	number of m-trails traversing node v_j
$\mathcal{A}^j$	the alarm code table seen at node $v_j \in V$
$\|\|\mathcal{T}\|\|_V$	normalized cover length, see 1.1

$\mathcal{T} = \{T_1, \ldots, T_b\}$ where $b = |\mathcal{T}|$ is the number of m-trails, and each node $v_j \in V$ can achieve L-UFL according to the on-off status of m-trails in $\mathcal{T}^j$ - the subset of $\mathcal{T}$ containing the m-trails passing through v_j. In this study we focus on node failures, and assume the links are perfect. Thus each m-trail T_i is a set of nodes that can be connected in G. Let $b_j = |\mathcal{T}^j|$ the number of m-trails seen at node v_j. Let $\mathcal{A}^j$ denote the alarm code table seen at node v_j, which is an $|V| \times b_j$ matrix with $a^j_{i,[k]} = 1$ if the k-th m-trail seen at v_j traverse node v_i and 0 otherwise. We say the failure of node v_i can be localized at node v_j if the i-th row of $\mathcal{A}^j$ is unique.

The set of m-trails $\mathcal{T}^j$ for v_j must satisfy the following two requirements:

(R1): Every node $v_i \in V$ should be passed by a unique set of m-trails in $\mathcal{T}^j$, such that every node has a unique alarm code seen by v_j.

(R2): T_i for $1 \leq i \leq b$ must be a set of nodes forming a connected subgraph of G.

The objective is to have minimal average number of m-trails seen at each node, i.e.

$$||\mathcal{T}||_V = \sum_{v_j \in V} \frac{b_j}{n} = \sum_{i=1}^{b} \frac{|T_i|}{n}, \tag{1.1}$$

where the equality holds because the total sum of the m-trails' lengths equals to the sum of the number of m-trails seen at all the vertices.

1.4 Related Work

L-UFL was first investigated in [2] where the goal was to determine one (or more) monitoring locations (MLs), that are nodes in the network where the m-trails can terminate, in order to collaboratively identify the failed SRLGs according to the alarms collected by the MLs. [10] extended this model by exploring the scenario where not only the terminating node but also an intermediate node of an m-trail can obtain its on-off status via optical signal tapping. An overview of fast failure localization with m-trails can be found in the book [9]. The study allocated m-trails which enable every nodes as L-UFL, called

Network-wide L-UFL (NL-UFL), capable for any single link failure. An efficient heuristic was developed for allocating m-trails in the shape of a spanning tree via link code swapping.

[11] is the only paper reported on node failures. It presents a heuristic algorithm (henceforward referred as GLS_{node}) that generates a random initial possibly invalid solution. Next through a series of minor modifications it is changed to valid solutions. To verify the validity of the solution, the L-UFL capability of each node is checked, which is at least $|SRLG| \cdot |V|$ steps. This function is launched hundreds of times, which ends up a rather time consuming job. In the present study we take a different approach, which ensures the validity of the solution by limiting the problem space to specific m-trail constructions. In Section 3.3 an extensive comparison of the proposed RMCA and the GLS_{node} algorithm is conducted. Also in [11], the authors determine a lower bound on the number of required monitoring trails (denoted with b) for NL-UFL for single node failures:

$$b \geq \lceil 1.62088 \log_2 (n) \rceil , \tag{1.2}$$

where n represents the number of nodes in the network. Note, that the cost of distributed failure localization is also affected by the length of the monitoring trails. Thus, in many studies ([2,6,13]) the normalized cover length of the set of m-trails is used as the objective function, that is the sum of the length of all m-trails. A length of a trail can be defined either as the number of links or as the number of nodes it traverses. In our study, we investigate single node failures, hence hereafter we refer to the number of traversed nodes by T_i as the length of the i-th trail.

Chapter 2

The Recursive Matching-Contraction Algorithm

It can be easily shown that if we assign unique binary codes with a length of k bits to the nodes in such a way, that at every bit position both the subgraph of the 1s and the 0s are connected, then NL-UNFL is achievable with $2k+1$ m-trails. Basically, the first k bits have to be the unique binary codes of the nodes, and then the complements of these binary codes have to be appended to them, which means $2k$ bits altogether. However, as we might have pairs of codes that are exactly the complements of each other, an additional trail has to be defined, which traverses all these complement pairs in the network. It guarantees NL-UNFL, as every node v receives the information of all the first k bits (which assumed to be a globally unique set of binary codes): either the original, i-th ($1 \leq i \leq k$) trail traverses v, or its complement pair $i+k$. The last trail ensures the recognition of the faultless state.

Assigning unique binary codes in a way mentioned above is not trivial as keeping the value of k as low as possible is one of our main objectives. A possible solution could be the following:

- Divide the given $G = (V, E)$ graph's vertices into two disjoint color classes $\mathcal{R}$ an $\mathcal{B}$ in such a way, that both classes compose a connected subgraph in G. Find a maximum matching $\mathcal{M}$ between $\mathcal{R}$ and $\mathcal{B}$. Vertices in the same classes are assigned the same binary value at the current bit position[1].
- Contract $\forall e \in \mathcal{M}$ resulting a new graph $G' = (V', E')$.
- Continue iterating these two steps on the new graphs, until only one vertex remains.

As for $\forall e = (i, j) \in \mathcal{M}$ the two vertices i and j are in different color classes (meaning they have different binary values in the given position), they can be distinghuised further on from each other. Thus, their contraction will not cease the uniqueness of the bit sequences, but it reduces the size of the graph, so that the algorithm has to work on a smaller graph at every iteration. It can be easily shown, that at the end of the algorithm, all the vertices

[1] E.g. vertex v is assigned 1, iff $v \in \mathcal{R}$ in the current iteration, and 0 otherwise.

are assigned a unique binary code.[2] Moreover, the subgraphs of both the 1s and the 0s are connected at each bit position, as we required the classes' connectivity at every iteration.

Obviously, the hardest part of the algorithm described above is the construction of the two disjoint subgraphs with maximum matching. Edmonds' algorithm, presented in [4], finds a maximum matching in a general graph in polynomial time, however, it does not take into account the color classes, and their connectivity, therefore it cannot be used for our case. A rather simple way to implement the abovementioned constrained matching is through Integer Linear Programming, which will be described in the next subsection.

The theoretical lower bound on the cover length for NL-UNFL is $\lceil \log_2 n \rceil$ as every node need to distinguish $n-1$ different node's failure, and the faultless state. The minimum $||\mathcal{T}||_V$ value, that the proposed Recursive Matching-Contraction Algorithm (RMCA) can reach is

$$||\mathcal{T}||_{V,min} = \lceil \log_2 n \rceil + \kappa, \tag{2.1}$$

where

$$\kappa = \frac{2 \cdot (n - 2^{\lceil \log n \rceil - 1})}{n}, 0 < \kappa \leq 1. \tag{2.2}$$

In order to achieve this value, the number of merging iteration steps should be $\lceil \log_2 n \rceil$, meaning, that after the $\lceil \log_2 n \rceil$-th step the contracted graph is composed of a single vertex. If we encode n nodes in $\lceil \log_2 n \rceil$ bits, then a minimum number of $n - 2^{\lceil \log n \rceil - 1}$ complement codes will be obtained, as there are $2^{\lceil \log n \rceil - 1}$ codes in the set of binary sequences with length of $\lceil \log_2 n \rceil$ that are pairwisely not complement of each other. Let δ_k denote the needed matching cardinality, and n_k the number of nodes in the k-th iteration. It can be easily shown, that if we would like to encode n nodes in $\lceil \log_2 n \rceil$ bits, then $\delta_k \geq n_k - 2^{\lceil \log_2 n \rceil - 1}$ should be satisfied for every k.

2.1 Integer Linear Program (ILP)

The mathematical tool of linear programming (LP) lets us optimize a certain objective function with respect to certain constraints formulated in inequalities. Integer LP implies another constraint: the defined variables can only take integer values. ILP is considered a useful tool for formulating combinatorical optimization problems.

Let y_e be a binary variable for $\forall e \in E$, which is 1 iff e is chosen to be in $\mathcal{M}$, and zero otherwise. The objective function is the sum of these y_e variables, as we would like to maximize the cardinality of $\mathcal{M}$. The constraints are the followings. For every vertex, the sum of its incident edges' y_e values should be no more than 1, as a matching is formed by a set of independent edges Eq. (2.4). Let the two color classes be $\mathcal{B}$ (blue) and $\mathcal{R}$ (red). The binary variable x_v^b is 1, iff v is blue, and similarly x_v^r is 1, iff v is red, and 0 otherwise. Obviously the sum of these two values for every v should be 1, as in Eq. (2.5). Eq. (2.6) guarantees that for all the edges in $\mathcal{M}$, the vertices incident to e are from different color classes. In order to ensure the color classes' connectivity, new variables have to be

[2] Two vertices can be contracted, iff they have different binary values at the current bit position, meaning they can be distinguished further on. This rule recursively holds for all the contractions, hence, when the last contraction occurs, all the nodes have unique codes.

introduced. Let us transform our initial graph G into a directed graph $G' = (V, E')$, so that instead of each undirected edge $e = (i, j)$, two directed arcs are presented; one from i to j, and another one from j to i. Assign two variables for each of these new arcs to represent blue and red flow values going through on the given arc: $z^b_{(i,j)}$ and $z^r_{(i,j)}$ respectively. The flow value can be larger than 0, only if both the source and the target vertices are from the same color class as the flow itself (e.g. $z^b_{(i,j)}$ can be positive, if both i and j are in the blue color class). β is a sufficiently large number, in this case $\beta = |V|$ is an appropriate choice Eq. (2.7). The sum of the incoming flow and the node's own color class variable (x^b_v and x^r_v) should be no more than the outgoing flow. This holds for both color classes and for all the vertices, except one from each class. The result will be a directed tree for both color classes, where the root is the abovementioned exception node. In order to satisfy the flow equations for the roots as well, another binary variable is introduced for every vertex for both classes; w^b_v and w^r_v are assigned a value of 1, iff v is the root of the blue or the red tree respectively Eq. (2.8). As there is exactly one root for each classes, and the root should have the same color as the flow itself, the inequalities of Eqs. (2.9) and (2.10) should be appended.

This formulation might seem a bit complicated at first sight, nevertheless we have described a graph theoretical optimization problem with only a few inequalities. Unfortunately, there is no known polynomial time algorithm for solving ILP problems, consequently the execution time increases excessively with the size of the input. Therefore, in the next section we try to devise a heuristic approach for this matching-maximizing problem.

$$\max . : \sum_{e \in E} y_e \tag{2.3}$$

s.t.:

$$\sum_{e \in N_G(v)} y_e \leq 1 \quad \forall v \in V \tag{2.4}$$

$$x^b_v + x^r_v = 1 \quad \forall v \in V \tag{2.5}$$

$$x^b_i + x^b_j \geq y_e \quad x^r_i + x^r_j \geq y_e \quad \forall e = (i, j) \in E \tag{2.6}$$

$$\begin{array}{ll} z^b_{(i,j)} \leq \beta \cdot x^b_i & z^r_{(i,j)} \leq \beta \cdot x^r_i \\ z^b_{(i,j)} \leq \beta \cdot x^b_j & z^r_{(i,j)} \leq \beta \cdot x^r_i \end{array} \quad \forall e = (i, j) \in E' \tag{2.7}$$

$$\begin{array}{l} x^b_i + \sum_j z_{(j,i)^b} \leq \beta ? w^b_i + \sum_j z^b_{(i,j)} \\ x^r_i + \sum_j z_{(j,i)^r} \leq \beta ? w^r_i + \sum_j z^r_{(i,j)} \end{array} \quad \forall i \in V \tag{2.8}$$

$$\begin{array}{l} \sum_{v \in V} w^b_v = 1 \\ \sum_{v \in V} w^r_v = 1 \end{array} \tag{2.9}$$

$$w^b_v \leq x^b_v \quad w^r_v \leq x^r_v \quad \forall v \in V \tag{2.10}$$

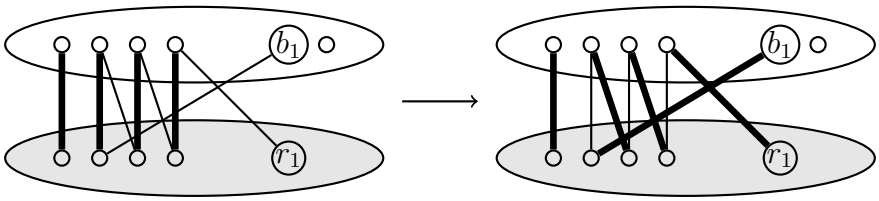


Figure 2.1: *Augmenting path in a bipartite graph*

2.2 Local Search Heuristic

The solution given by the ILP is optimal for the abovementioned maximal matching problem, however, the computing time grows immensely with the size of the input graph. A promising, fast alternative of the ILP can be a simple local search heuristic, that tries to maximize the cardinality of an initial matching by means of certain augmenting paths. Hereafter, RMCA will refer to the algorithm that utilizes the local search heuristic described in this section for matching cardinality maximizing.

In case of bipartite graphs the concept of an augmenting path is well defined: it is an alternating path from u to v, where u and v are two unpaired vertices in different color classes, thus, neither of them are included in the current matching. An alternating path is a sequence of edges where every second edge is included in the current matching set $\mathcal{M}$. Fig. 2.1 shows an example of an augmenting path in a bipartite graph. The white and grey ellipses represent the two color classes. Obviously, this definition of augmenting path can be used for general graphs as well; if we define the two color classes, augmenting paths can be found regardless the edges that connect vertices in the same color class. Hence, the concept of augmenting path (AP) is implemented in the heuristic.

2.2.1 Initial coloring

First of all, the two color classes should be defined; this is achieved by random walks initiated from two nodes r_0 and b_0. The two classes spread in a step-by-step manner, if one of the classes has no uncolored neighbour, then the rest of the graph is colored by the other class. If the graph is connected (as we always assume), the walks will eventually cover all the nodes in the network, classifying them into two color classes, $\mathcal{R}$ and $\mathcal{B}$ respectively. Then, a maximal matching $\mathcal{M}_0$ is found in a greedy way, where an independent set of inter-class edges is chosen. Handling G as a bipartite graph, a maximum matching $\mathcal{M}$ can be obtained from $\mathcal{M}_0$ by means of APs. However, the aforementioned intra-class edges (which do not exist in bipartite graphs) could be utilized as well, in order to define new types of augmenting paths, hereafter let us refer to them as mixed augmenting paths (MAPs). In the following subsections I propose three types of MAPs, and illustrate them with examples taken from the simulations conducted on some real network topologies. In particular, I demonstrate all the three MAPs on the 16-node Pan-European network. The figures show both the original network, and the rearranged graph, where the nodes belonging the same color class are grouped in the same ellipse. The two ellipses represent the two color classes, and the nodes' colors also indicate the class they belong to. A visual summary of the three

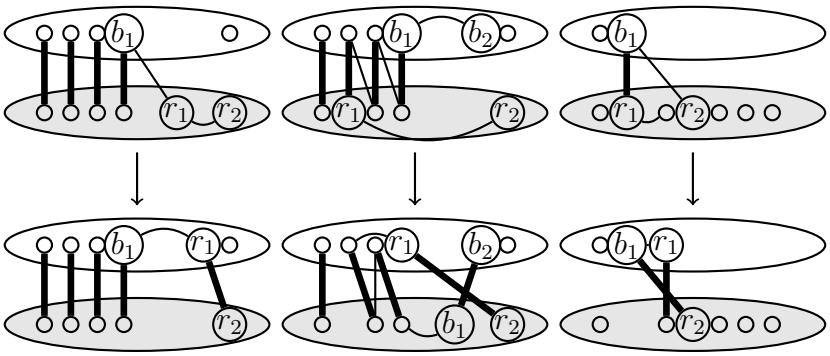


Figure 2.2: *An example of the 3 different MAPs*

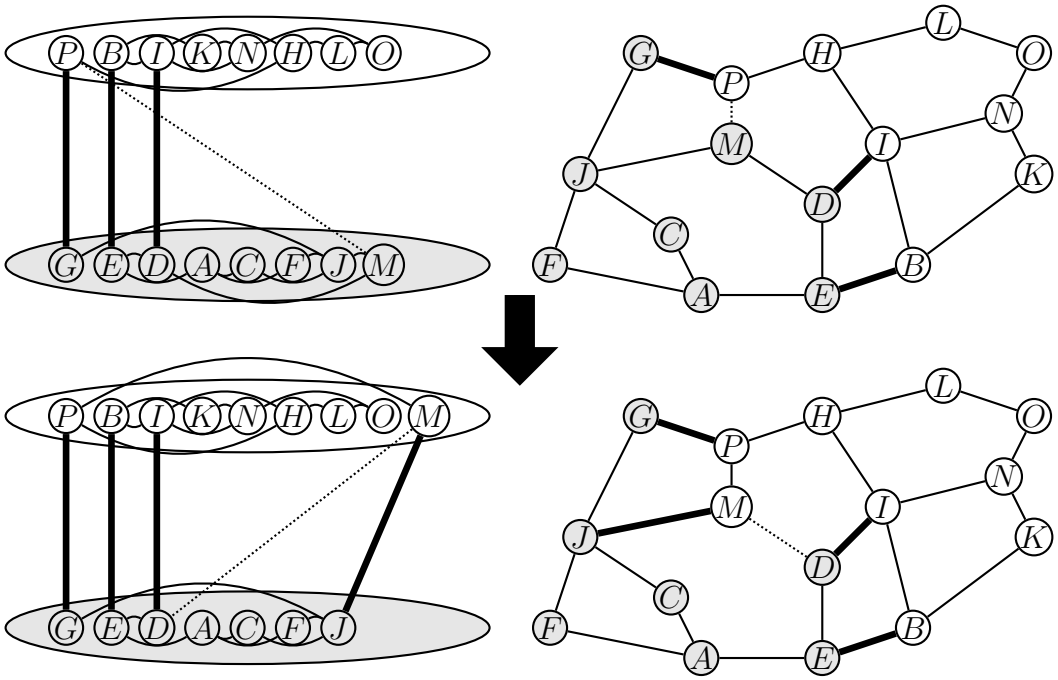


Figure 2.3: *An example of MAP type 1 on the Pan-European network*

MAPs is shown on Figure 2.2. Note, that in this case, not all the edges are included on the Figure 2.2, only the ones that are directly utilized during the augmenting method and the ones included in $\mathcal{M}$ (the bold edges).

2.2.2 MAP type 1

A very simple way to increase the cardinality of $\mathcal{M}$ is the following; supposing that there is an unpaired vertex $r_1 \in \mathcal{R}$, that is connected to an unpaired vertex $r_2 \in R$ and another vertex $b_1 \in \mathcal{B}$, r_1 can be transposed to the color class $\mathcal{B}$, and the edge $e = (r_1, r_2)$ can be added to $\mathcal{M}$ (see Figure 2.2). In order to keep the color classes' connectivity another very important constraint has to be satisfied: r_1 must not be an articulation point (cut vertex) of its original color class $\mathcal{R}$; meaning that its removal must not affect the connectivity of $\mathcal{R}$.

We can follow this procedure on the real network example, shown on Figure 2.3. Initially, $\mathcal{M}$ contains three edges, namely $G - P$, $D - I$ and $E - B$, marked with bold edges. All

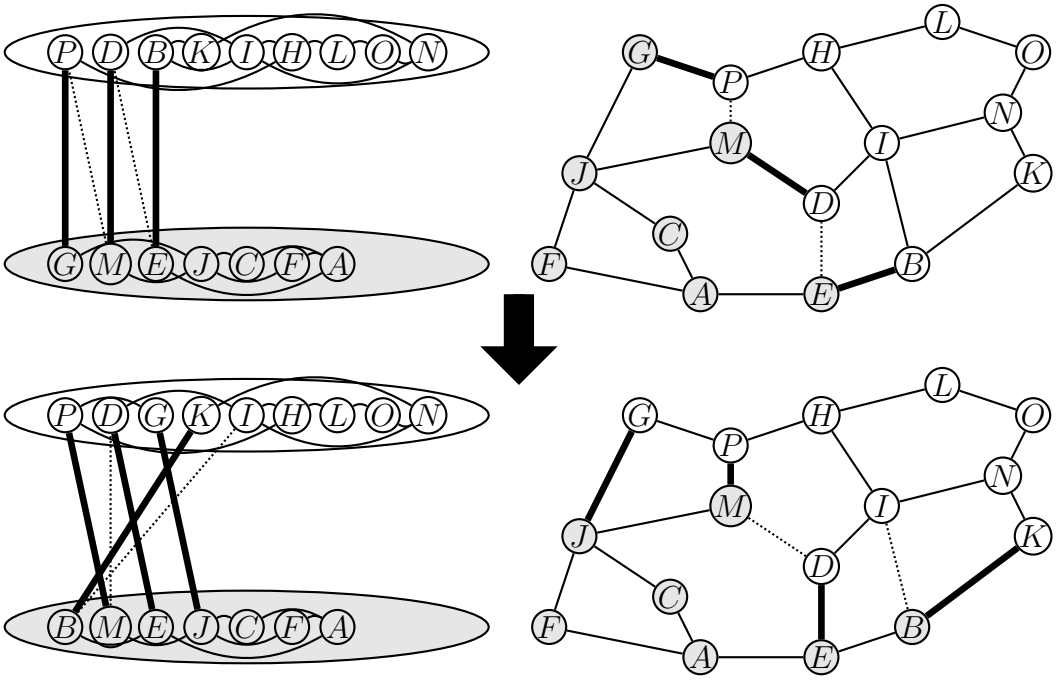


Figure 2.4: *An example of MAP type 2 on the Pan-European network*

the remaining inter-class edges are denoted by dotted lines. In our case, there is only one inter-class edge, namely the one connecting M and P. As we have seen, the execution of MAP type 1 requires an unpaired vertex that is connected both to a vertex from the other color class and another unpaired one from the same class. Fortunately, M satisfies these criteria (through the edges $M-P$ and $M-J$), furthermore, it is not an articulation point in its own class, therefore the augmenting step can be executed. Hence, M is transposed to the other class, and $M-J$ is added to $\mathcal{M}$ increasing its cardinality to 4. As color classes changed, the previous inter-class edge $M-P$ becomes an intra-class edge, while a new inter-class edges between M and D appears. Note, that further MAP type 1 steps cannot be performed, since this single inter-class edge connects two vertices that are already paired. However, further MAPs can be defined in order to increase the cardinality of $\mathcal{M}$.

2.2.3 MAP type 2

Another possible matching-increasing method is very similar to the augmenting path for bipartite graph matchings. Provided that there is an alternating path between two paired vertices $r_1 \in \mathcal{R}$ and $b_1 \in \mathcal{B}$, and both r_1 and b_1 are connected to an unpaired vertex in their own color class (r_2 and b_2 respectively), then an augmenting path is found (see Fig. 2.2). r_1 is transposed to $\mathcal{B}$, and b_1 is moved to $\mathcal{R}$. The new matching $\mathcal{M}'$ is composed of those inter-class edges of the augmenting path, that were not included in the initial matching $\mathcal{M}$ (the ones denoted with dotted lines) and two new edges r_1-r_2 and b_1-b_2. Note, however, that neither r_1 or b_1 can be cut vertices in their own class, since the class-connectivity constraints have to be satisfied.

On Fig. 2.4 an initial matching of three edges is shown: $G-P$, $M-D$ and $E-B$. Besides these three paired edges, there are two inter-class edges, $P-M$ and $D-E$. Furthermore,

this five edges construct an alternating path from G to B: $G - P - M - D - E - B$. As G and B are not cut vertices in their color classes, and they both connected to an unpaired vertex of the same class (J and K respectively), we can use the augmenting path described above. Thus, the resulting matching $\mathcal{M}'$ is composed of 4 edges: $J - G$, $M - P$, $E - D$ and $B - K$. Note, that we could have chosen node I to be the new pair of B, as I had also been unpaired before the augmenting step. Whenever the algorithm reaches a state like this, i.e. it has to choose between two or more seemingly equivalent options, it selects randomly one of them. Although, choosing $B - K$ instead of $B - I$ into $\mathcal{M}'$ in this particular situation is disadvantageous for the following reason. The resulting graph does not contain any subgraphs that satisfies the criteria for either MAP type 1 or type 2, therefore no further augmenting steps can be done. [3] On the other hand, if node I had been chosen to be the new pair of B, then through node K an additional MAP type 1 could have been defined, increasing the matching size to 5. As the investigation of all the possible outcomes would require significant computing time, for the sake of simplicity and fastness the algorithm chooses randomly in these scenarios.

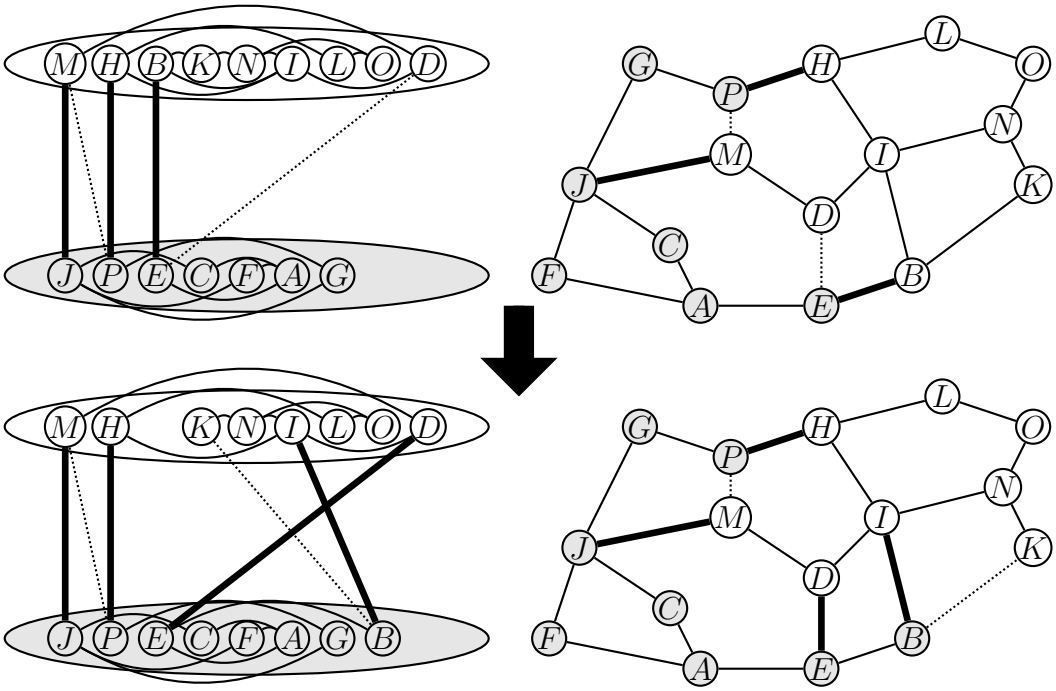


Figure 2.5: *An example of Spreading and MAP type 3 on the Pan-European network*

2.2.4 Spreading + MAP type 3

It might happen that after the initialization, one color class is significantly larger, than the other one. In that case, the augmenting process gets stuck after a few steps, and the

[3] One might think, that I is an appropriate choice for MAP type 1 through edges $B - I$ and $I - N$. However, I is a cut vertex in its class, thus if it was transposed to the other class, D would become separated from the rest of its class. The path $A - E - D - M - P - H$ for MAP type 2 also violates the cut vertex constraint at E and P.

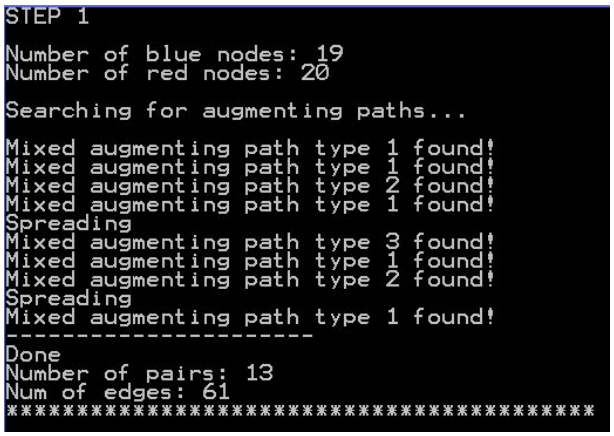


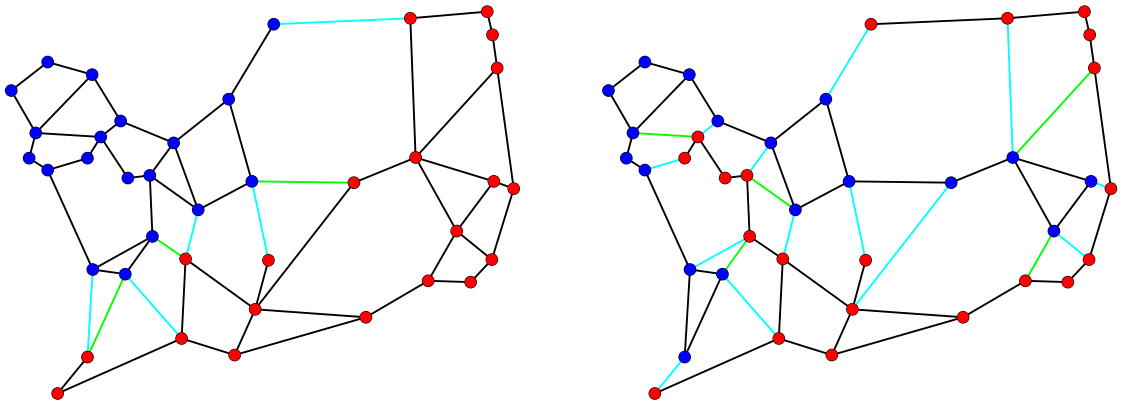
Figure 2.6: *Console window of the first step of RMCA on the North American network*

resulting matching will probably have a suboptimal cardinality. A possible way to overcome this issue is to somehow let the smaller class "spread" into the larger one. Fig. 2.2 shows one step of spreading. If the larger color class contains a paired vertex r_1 (with pair b_1) that is not a cut vertex, and is connected to a not paired vertex r_2 within its class, then it can be moved to the smaller class, adding the edge $r_1 - r_2$ to $\mathcal{M}$ and removing $r_1 - b_1$. The algorithm seeks spreading possibilities whenever it cannot find further MAPs of type 1 and type 2, and if one class is larger than the other one. Spreading does not increase the cardinality of $\mathcal{M}$, it only balances the number of the vertices in the two color classes. However, if the pair of r_1, b_1 is connected to an unpaired vertex $r_2 \in \mathcal{R}$, then that edge $e_1 = (b_1, r_2)$ can be added to $\mathcal{M}$. Thus, spreading might increase the cardinality of $\mathcal{M}$ with this modification. This method is hereafter referred to as the 3^{rd} type of MAPs.

A MAP type 3 step is demonstrated on Fig. 2.5. In the initial configuration no more type 1 or type 2 MAPs can be found. Nevertheless, B is a promising candidate for being r_1 in MAP type 3, as it is a not cut vertex paired with E, and it has two unpaired neighbours, I and K. Hence, MAP type 3 can be executed, and as seen before, the algorithm chooses randomly between the two suitable edges, $B - I$ and $B - K$. Fortunately, $B - I$ has been chosen to $\mathcal{M}'$ this time, so that a further MAP type 1 can be performed through the edge $B - K$ and $K - N$.

Fig. 2.6 shows the algorithm's output in the console window during its execution on the 39-node North American optical network. 19 blue, and 20 red nodes are present in the network after the initial coloring. Thereafter, the augmenting process is initiated, and a total number of 8 matching cardinality-increasing augmenting steps are performed (along with two spreading steps). The resulting matching is composed of 13 pairs, which is a quite high number regarding the network's low average nodal degree (3.13). The initial, and the augmented matchings are shown on Figure 2.7 (a) and (b). Pale blue edges represent the edges that are included in the matching set, while green edges are the remaining inter-class edges.

Obviously, it is possible to define further MAPs, that might increase the cardinality, but for the sake of simplicity and quickness I have used only the ones introduced above.



(a) Initial matching **(b)** Augmented matching

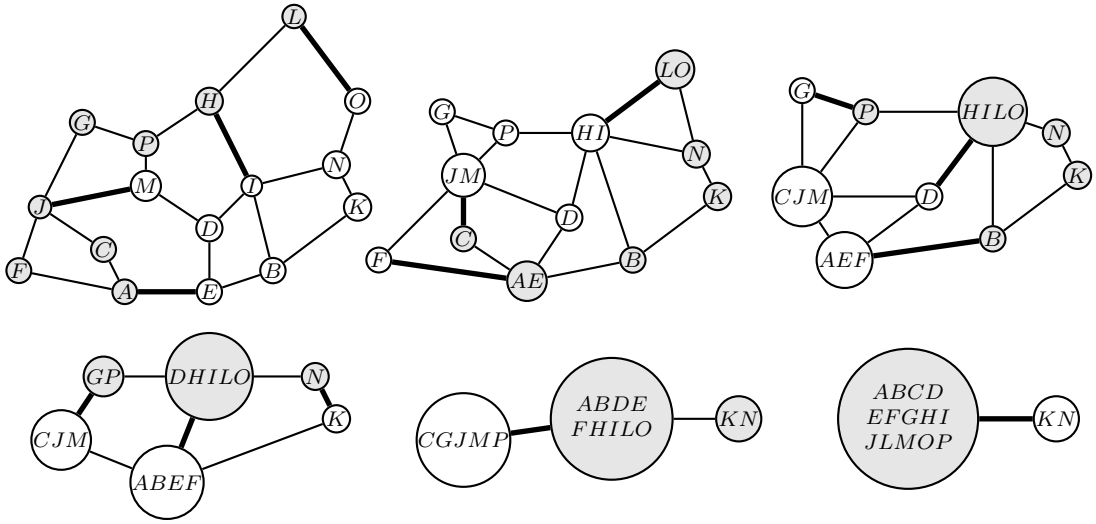
Figure 2.7: *39-node*

2.2.5 Illustrative example

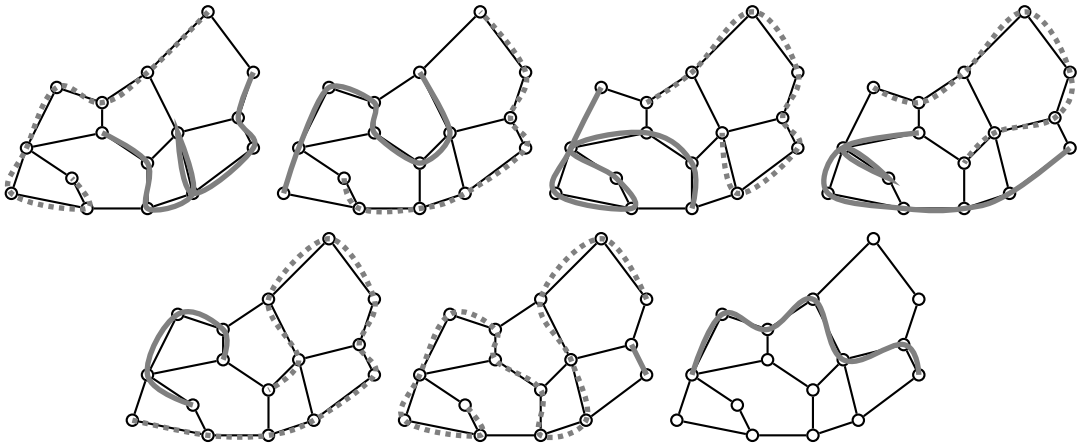
Fig. 2.8 shows the sequence of steps and the resulting m-trails of RMCA on the 16-node Pan-European optical network. Each merging step defines two new m-trails representing the two disjoint color classes, colored with white and grey. On Fig. 2.8(a) the bold edges at every step are the ones that will be contracted for the next iteration, so that their two vertices' labels can be merged together, indicating, that they will be on the same m-trails further on. Thus, on the initial graph the edges connecting $J-M$, $A-E$, $H-I$ and $L-O$ can be contracted. Consequently, in the next iteration we are looking for a matching with connected color classes in a smaller, 12-node graph. The color classes and the obtained matchings are the result of the previously proposed graph coloring, and augmenting methods. The same matching-contraction steps can be executed in the next iterations, each step defines two m-trails. The importance of the connectivity of both color classes is shown by Fig. 2.8(b). The corresponding two m-trails for the two colors classes are represented with a solid and a dashed line (for white and grey classes respectively) for each step, resulting in 12 trails altogether.

As mentioned before, an additional trail should be added to the solution when we have nodes that are assigned complement binary codes, as they cannot take notice of each other's failure. On our example K-G, and N-J are complement pairs (meaning they are on different trails for all the 6 pairs), so that the last m-trail has to include them along with vertices H, I and P to make the trail connected. In our case T_{comp} will increase $||\mathcal{T}||_V$ by $\frac{7}{16}$, as it traverses 7 vertices in the 16-node network. The resulting Alarm Code Table (ACT) is shown on Table 2.1, $\{T_1, \ldots, T_{13}\}$ are the 13 m-trails. A node is included in the i-th m-trail, iff the i-th bit in the corresponding row is 1. Note, that $\{T_6, \ldots, T_{12}\} = \{\overline{T_1}, \ldots, \overline{T_6}\}$, where $\overline{T_i}$ is the bitwise complement of T_i. Let's denote the first 12 trails by $\{T_{M1}, \ldots, T_{M6}, \overline{T_{M1}}, \ldots, \overline{T_{M6}}\}$, where M refers to the Matching that basically generated these trails. Also, T_{comp} denotes the last, additional trail, referring to the *comp*lement pairs that it contains.

The distinction of certain categories of m-trails is not necessary, however, it might be



(a) The graph after merging in each step



(b) The trails

Figure 2.8: *An illustrative example of the algorithm on 16-node European reference network.*

helpful, as a new type of m-trails will be presented in the next subsection.

2.2.6 Articulation point removal

As each matching-contraction step increases the number of m-trails by 2 (and $||\mathcal{T}||_V$ by 1), the main target is to keep the contraction steps as low as possible. The proposed RMCA performs very well on graphs with large average nodal degree, nevertheless, it has some limitations on network topologies with lower nodal degrees. The main reason for this behaviour is the emergence of cut vertices during edge contractions. Once the algorithm reaches a point, where an articulation point appears, the convergence becomes slower, much lower number of edges can be contracted at each step. The underlying reason for this deceleration is the following: a cut vertex v seperates the graph into two components A and B, thus when the color classes are being constructed, vertices in one of the components have to be in the same color class as v is. Consequently, no edges in either A or B can be included in the matching. Thus, the upper bound for $|\mathcal{M}|$ is $max\{\lfloor|A|/2\rfloor, \lfloor|B|/2\rfloor\}$, that

Table 2.1: *ACT for the 16-node European network*

Nodes \ M-trails	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}
	T_{M1}	T_{M2}	T_{M3}	T_{M4}	T_{M5}	T_{M6}	$\overline{T_{M1}}$	$\overline{T_{M2}}$	$\overline{T_{M3}}$	$\overline{T_{M4}}$	$\overline{T_{M5}}$	$\overline{T_{M6}}$	T_{comp}
A	0	0	1	1	0	0	1	1	0	0	1	1	0
B	1	0	0	1	0	0	0	1	1	0	1	1	0
C	0	0	1	1	1	0	1	1	0	0	0	1	0
D	1	1	1	0	0	0	0	0	0	1	1	1	0
E	1	0	1	1	0	0	0	1	0	0	1	1	0
F	0	1	1	1	0	0	1	0	0	0	1	1	0
G	0	1	1	0	1	0	1	0	0	1	0	1	1
H	0	1	0	0	0	0	1	0	1	1	1	1	1
I	1	1	0	0	0	0	0	0	1	1	1	1	1
J	0	1	1	1	1	0	1	0	0	0	0	1	1
K	1	0	0	1	0	1	0	1	1	0	1	0	1
L	0	0	0	0	0	0	1	1	1	1	1	1	0
M	1	1	1	1	1	0	0	0	0	0	0	1	0
N	1	0	0	0	0	1	0	1	1	1	1	0	1
O	1	0	0	0	0	0	0	1	1	1	1	1	0
P	0	1	0	0	1	0	1	0	1	1	0	1	1

is usually way lower than the theoretical optimum for a graph containing no cut vertices.

A quite intuitive way to overcome this issue is the following. If we unambiguously identify the cut vertex v with a few additional m-trails, then all the further trails can go through v. Therefore, v can be removed from the graph, and all of its neighbour vertices can be connected with each other (as they are linked through v). Figure shows an articulation point removal step on a random planar graph.

The only remaining question is the way we identify the cut vertex for all the nodes in the network. It is easy to see, that adding four appropriate trails is always sufficient: two for the two components containing the cut vertex, and two more monitoring trails for the two components without the cut vertex. The drawback for this vertex removal is twofold. First, and most importantly it will increase the normalized cover length value by two, as every node is crossed by two out of the abovementioned four additional trails. Secondly, as the removed cut vertex contained several nodes from the original graph (because of the contractions of the previous steps), its removal means that these vertices might be included in both color classes during the forthcoming iterations, which will also slightly increase the cover length. However a large number of pairs can be found in the next iterations thanks to the newly added edges, which will essentially improve the RMCA algorithm in terms of $||\mathcal{T}||_V$.

The four new m-trails are further denoted with $\{T_{A1}, \ldots, T_{A4}\}$, where A refers to the *A*rticulation point removal. T_{A1} and T_{A2} are the trails going through component A without and with the cut vertex, respectively, while T_{A3} and T_{A4} are the trails traversing the other component, B. A node a in component A can unambiguously identify the failure of the cut vertex v, as in this case T_{A1} would be faultless while T_{A2} would indicate an error. The same holds for nodes in component B.

Leaf vertices (nodes with degree one) are managed in a different way. Instead of removing their neighbour cut vertex and adding four more m-trails as it was described above, we can simply contract them to their neighbour, by defining 2 new m-trails for every leaf node v: one containing only v, and another one containing the rest of the graph: $G \setminus \{v\}$. Basically,

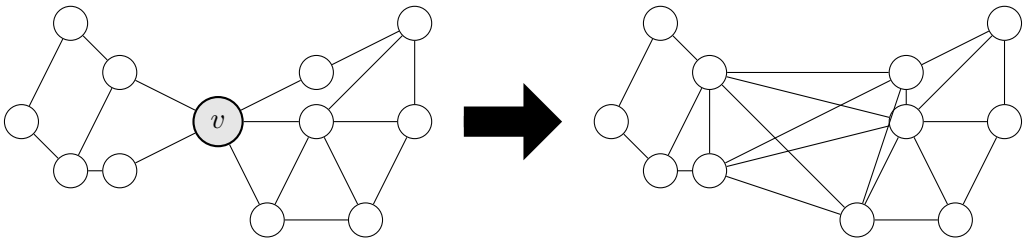


Figure 2.9: *Example of articulation point removal on a random planar graph*

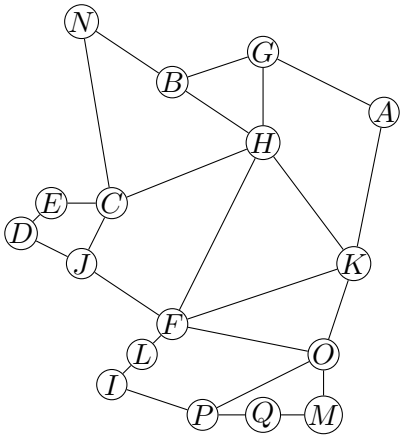


Figure 2.10: *17-node German optical network*

this is exactly the same mechanism, that the algorithm does during matching contraction, so the newly constructed 2 trails can be placed in the set of T_M trails.

2.2.7 Unnecessary m-trail removal

Following the previously described steps clearly guarantees NL-UNFL, however, it might happen that a few m-trails become superfluous, meaning they can be removed from our set $\mathcal{T}$ without violating the unambiguousity of the localization, thus a decrease in terms of $||\mathcal{T}||_V$ can be achieved. Superfluous trails are particularly likely to appear during articulation point removals, where four trails are added to $\mathcal{T}$ for each vertex removal, as seen in Subsection 2.2.6. Nevertheless, while adding four trails ensures further localization unambiguousity for all possible scenarios, most of the times two trails are sufficient in practice.

Therefore, as a final step, RMCA eliminates all the unnecessary monitoring trails from $\mathcal{T}$. Basically, a subroutine loops through the trails one by one, and if the i-th trail is needless (i.e. the ACT provides NL-UNFL without the i-th trail), the i-th column is removed from the ACT. An example of the m-trail reduction process is shown on Table 2.2(a), that includes the ACT after RMCA's execution on the 17-node German optical network (before m-trail elminiation), and the final ACT after superfluous m-trail elimination (Table 2.2(b)). Observe, that in this case, no T_{comp} was needed, as $\mathcal{T}$ did not contain any complementer pairs. Although, the ACT does contain T_A trails, as a cut vertex was removed. Note, that an evolved cut vertex always contains more than one node from the original graph (in a 2-connected graph), because of the algorithm's contracting nature. The removed articulation

Table 2.2: *ACT for the 17-node German network before and after m-trail elimination*

(a) Before elimination

	T_{M1}	T_{M2}	T_{M3}	T_{M4}	T_{M5}	T_{M6}	T_{M7}	$\overline{T_{M1}}$	$\overline{T_{M2}}$	$\overline{T_{M3}}$	$\overline{T_{M4}}$	$\overline{T_{M5}}$	$\overline{T_{M6}}$	$\overline{T_{M7}}$	T_{A1}	T_{A2}	T_{A3}	T_{A4}
A	0	0	1	0	0	1	1	1	1	0	1	1	0	0	0	0	1	1
B	0	0	1	1	0	1	1	1	1	0	0	1	0	0	0	0	1	1
C	0	0	0	1	0	0	0	1	1	1	0	1	1	1	0	0	1	1
D	1	1	0	1	0	0	0	0	0	1	0	1	1	1	0	0	1	1
E	0	1	0	1	0	0	0	1	0	1	0	1	1	1	0	0	1	1
F	1	0	1	1	0	1	1	0	1	1	1	1	1	1	0	1	0	1
G	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	1
H	1	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	1
I	1	0	0	0	0	0	1	0	1	1	1	1	1	0	1	1	0	0
J	1	0	0	1	0	0	0	0	1	1	0	1	1	1	0	0	1	1
K	0	0	1	1	0	1	1	1	1	1	1	1	1	1	0	1	0	1
L	1	0	0	0	0	1	1	0	1	1	1	1	0	0	1	1	0	0
M	1	0	1	1	0	0	0	0	1	0	0	1	1	1	1	1	0	0
N	0	0	0	1	0	1	1	1	1	1	0	1	0	0	0	0	1	1
O	0	0	1	1	0	0	0	1	1	0	0	1	1	1	1	1	0	0
P	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0
Q	1	0	1	0	0	0	0	0	1	0	1	1	1	1	1	1	0	0

(b) After elimination

	T_{M1}	T_{M2}	T_{M3}	T_{M4}	T_{M6}	T_{M7}	$\overline{T_{M1}}$	$\overline{T_{M2}}$	$\overline{T_{M3}}$	$\overline{T_{M4}}$	$\overline{T_{M5}}$	$\overline{T_{M6}}$	$\overline{T_{M7}}$	T_{A1}	T_{A3}
A	0	0	1	0	1	1	1	1	0	1	1	0	0	0	1
B	0	0	1	1	1	1	1	1	0	0	1	0	0	0	1
C	0	0	0	1	0	0	1	1	1	0	1	1	1	0	1
D	1	1	0	1	0	0	0	0	1	0	1	1	1	0	1
E	0	1	0	1	0	0	1	0	1	0	1	1	1	0	1
F	1	0	1	1	1	1	0	1	1	1	1	1	1	0	0
G	0	0	1	1	1	1	1	1	0	0	0	0	0	0	1
H	1	0	1	0	1	1	0	1	0	1	1	0	0	0	1
I	1	0	0	0	0	1	0	1	1	1	1	1	0	1	0
J	1	0	0	1	0	0	0	1	1	0	1	1	1	0	1
K	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0
L	1	0	0	0	1	1	0	1	1	1	1	0	0	1	0
M	1	0	1	1	0	0	0	1	0	0	1	1	1	1	0
N	0	0	0	1	1	1	1	1	1	0	1	0	0	0	1
O	0	0	1	1	0	0	1	1	0	0	1	1	1	1	0
P	1	0	0	0	0	0	0	1	1	1	1	1	1	1	0
Q	1	0	1	0	0	0	0	1	0	1	1	1	1	1	0

point contained nodes F and K. Obviously, $\{F, K\}$ is a cutter subset in the original network (see Figure 2.10)

2.2.8 Pseudocode

A pseudocode of the RMCA algorithm is provided on Algorithm 1. Basically, the *while* cycle implies the recursive contracting behaviour of the algorithm. First, the eliminiation of the cut vertices take place if there is any. As it was discussed in Subsec. 2.2.6, four - T_A type of - trails are added to the current trail set, $\mathcal{T}$. The original graph's nodes (i.e. the real nodes of the network) contained by the removed vertices are restored in a register. Once they are removed from the graph, the forthcoming iterations do not add any bits to these nodes' codes. Hence, filling these bit positions for the articulation points occurs after the *while* cycle (Step 18). If the cut vertex is needed to make the i-th trail connected, than 1 is written to its i-th position. Therefore, if a node is removed in the k-th iteration, its binary value on trails $\{T_{k+1}, \ldots, T_l, \overline{T_{k+1}}, \ldots, \overline{T_l}\}$ is decided after the *while* cycle (l denotes the number of contracting steps, i.e. the number of iterations). Hence, the previously seen complement rule (i.e. T_{Mi} is 1, iff $\overline{T_{Mi}}$ is 0) does not necessarily hold for these nodes. For a simple example, consider the codes of nodes F and K on the previous ACTs of the German network (Table 2.2).

After the cut vertex removal, the graph is getting colored by the simple coloring pro-

cedure discussed in Subsec. 2.2.1, which provides two color classes $\mathcal{R}$ and $\mathcal{B}$ (as all the nodes are colored, and each of them is assigned only one color, $|\mathcal{R}| + |\mathcal{B}| = |V|$ holds). Since the color classes (and the corresponding inter-, and intra-class edges) are defined, an initial matching $\mathcal{M}$ can be established. Thereafter, the $|\mathcal{M}|$-increasing augmenting steps (AP, MAP type 1,2,3 and Spreading) are coming, until the current matching-size cannot be increased. Steps (11)-(15) show the binary code assignment of the nodes. As the graph's vertices gradually "grow" during the iterations (meaning they contain more and more nodes from the original network, see Fig. 2.8(a)), the algorithm goes through all the nodes included in a colored vertex v, and assigns 1 to the nodes if v is in $\mathcal{R}$, and 0 otherwise. Then, the edges that are included in the final matching are contracted, and the procedure starts again on a smaller graph, until only 1 vertex remains. After the cycle terminates, the cut vertices' codes are completed. If there are any complement codes in $\mathcal{T}$ (Step (20)), then T_{comp} is added (Step (21)). Finally, the superfluous trails are being removed (Step (22)).

Algorithm 1: RMCA

Input: Undirected graph $G = (V, E)$
Output: Set of monitoring trails,$\mathcal{T}$, that provides NL-UNFL

```
1  begin
2      It=0;
3      while |V|>1 do
4          ++It;
5          Eliminate cut vertices, adding 4 trails to T for each removed vertex;
6          Initial coloring of G, resulting R and B;
7          Greedy search for initial matching, resulting M;
8          repeat
9              Search for AP, MAP1, MAP2, Spreading and MAP3;
10             Update R, B, M;
11         until no more augmenting possibilities;
12         for ∀v ∈ V do
13             if v ∈ R then
14                 for ∀ node n in v do T^n_It=1 and T̄^n_It=0 ;
15             else
16                 for ∀ node n in v do T^n_It=0 and T̄^n_It=1 ;
17             end
18         end
19         for ∀e = (u, v) ∈ M do  contract e ;
20         Update V and E;
21     end
22     Complete T in cut vertices' rows;
23     Search for complement codes;
24     if T_comp is needed then append T_comp to T;
25     Delete unnecessary trails;
26 end
```

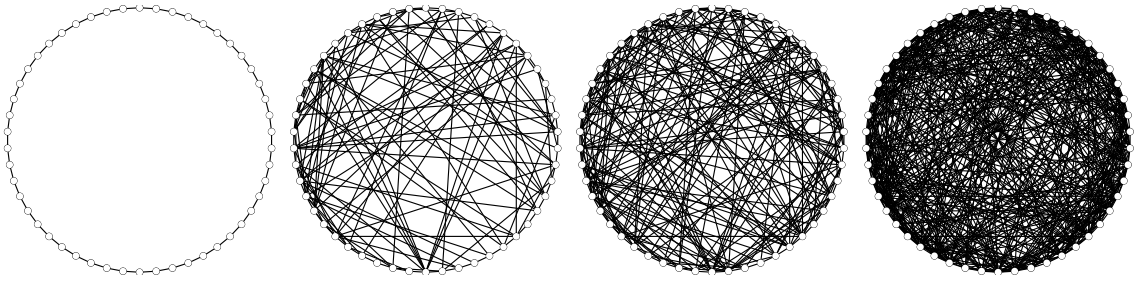
Chapter 3

Simulation

The development of an algorithm is usually an iterative process. Simulations can reveal a heuristic's weaknesses, that may lead to minor or major modifications on the original algorithm. This certainly holds for RMCA, as the concept of Spreading (Subsec. 2.2.4), Articulation point removal (Subsec. 2.2.6) and Superfluous trail elimination (Subsec. 2.2.7) all emerged during initial simulations on test graphs. Therefore, it is essential for a network planning algorithm to evaluate it on a large, diverse set of topologies. The proposed RMCA algorithm was tested on a large number of different input topologies including graphs representing real optical networks. RMCA was implemented in C++, with the usage of the LEMON graph library [1].

3.1 Erdős-Rényi graphs

At first, in order to verify the algorithm's correctness, fastness and scalability, a number of test graphs were generated with a slightly different version of the well-known Erdős-Rényi model. Let us denote the generated graph with $G = (n, m)$, where n and m are the number of nodes and edges in the graph, respectively. When investigating optical layer restoration, we are mainly interested in 2-connected topologies, as if the network is not 2-connected, the failure of a cut vertex v would separate the graph into two components, so that the traffic passing through v cannot be served. Therefore, instead of generating an entirely random graph with n nodes and m edges (as it is in the model), a ring of n nodes was



(a) Nodal deg. 2.0 **(b)** Nodal deg. 6.0 **(c)** Nodal deg. 10.0 **(d)** Nodal deg. 20.0

Figure 3.1: *Erdős-Rényi graphs with 50 nodes.*

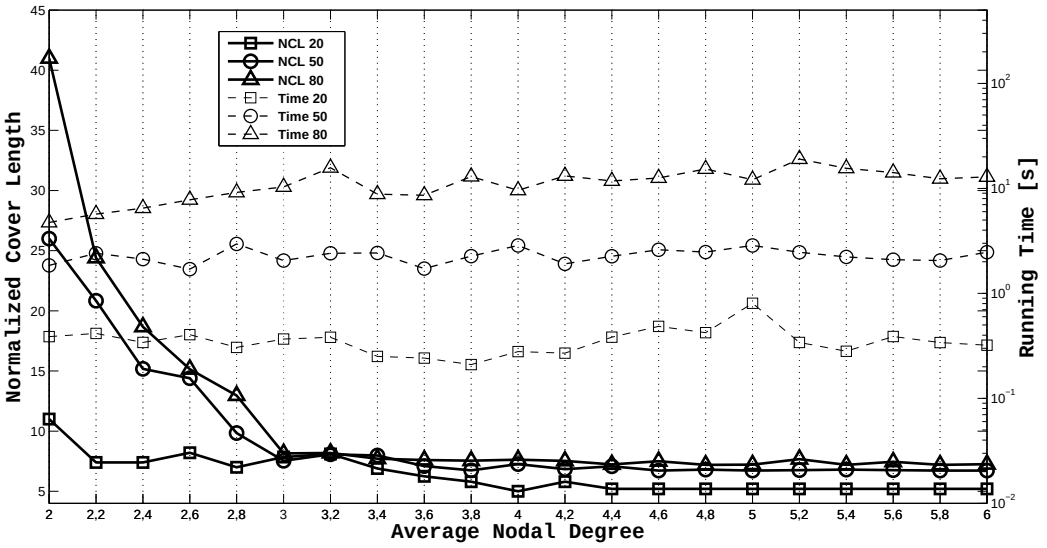


Figure 3.2: *RMCA's performance on Erdős-Rényi graphs of 20,50 and 80 nodes.*

generated initially, in order to ensure the graph's 2-connectivity, thereafter the remaining $m-n$ edges were added randomly. Figure 3.1 illustrates 50-node Erdős-Rényi graphs with different average nodal degrees.

With the aim of proving RMCA's correctness via a series of simulations, at first I have generated a number of Erdős-Rényi graphs with average nodal degree from 2.0 to 6.0 by 0.2 steps, for 3 different sizes: n=20, 50 and 80 nodes. RMCA was executed on that overall 63 graphs, and the results are shown on Fig. 3.2. Dashed lines represent the measured execution times as a function of average nodal degree, while solid lines denote the objective function value (i.e. the normalized cover length: $||\mathcal{T}||_V$) obtained by RMCA on the corresponding graph. The theoretical lower bounds on $||\mathcal{T}||_V$ for 20, 50 and 80 nodes are 5, 6 and 7 respectively. The results provided by RMCA clearly approach these bounds rapidly: at nodal degree 3.0 all the 3 lines are within a very small distance from their lowest possible values. The running time results are also promising: RMCA executed even on the largest, 80-node graphs within 15-20 seconds, which is a very short time interval from a network planning tool. Note, however, that the running time increases significantly with the input size, while the average nodal degree of the graph does not really affect it.

Another simple simulation was executed on Erdős-Rényi graphs in order to further investigate the scalability of RMCA. As mentioned before, we seek after solutions that have a logarithmic nature, meaning that the normalized cover length or the number of m-trails is not linear, but logarithmic function of the input size. Figure 3.3 shows the results of RMCA on some Erdős-Rényi graphs with sizes ranging from 10 to 120, and nodal degree from 2.6 to 4.2 with steps of 0.4. Obviously, the performance progresses with the increase of the average nodal degree; the more edges mean larger initial matching, and more possibilities for RMCA to enlarge the found matching. It can be observed, that the results of RMCA in terms of $||\mathcal{T}||_V$ are growing in a logarithmic fashion with the size of

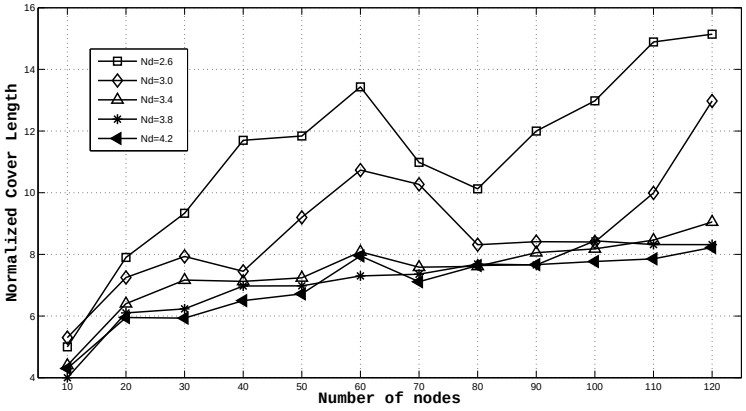
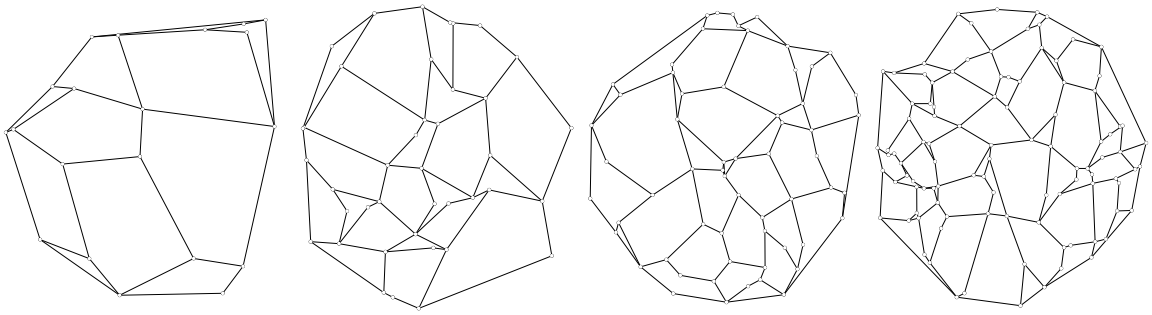


Figure 3.3: *RMCA's results on some Erdős-Rényi graphs*

the input, especially with higher nodal degrees.

3.2 Random planar graphs

Erdős-Rényi graphs, however, are scarcely used for modeling optical backbone networks, as real life topologies are seldom constructed in a random manner. Hence, instead of entirely random graphs, we should rather focus our attention on planar graphs. Planar means, that the graph can be embedded in the plane in such a way that no edges cross each other. I have generated a number of planar graphs with large diameter with a random planar graph generator [1] for creating optical network-like topologies. Figure 3.4 shows four example graphs with average degree of 3.0, created by the planar graph generator.



(a) n=20,$|E|$=30 **(b)** n=40,$|E|$=60 **(c)** n=60,$|E|$=90 **(d)** n=100,$|E|$=150

Figure 3.4: *Random planar graphs with average nodal degree 3.0.*

The average nodal degree of a 50-node graph has been increased by 0.08 steps from 2.0 to 8.0, at every step 5 different graphs were generated, resulting in a total number of 380 topologies. The algorithm was executed on each graph for 5 times. The mean $||\mathcal{T}||_V$ value and standard deviation for a given nodal degree were calculated from the corresponding 25 results. The algorithm was executed both with and without the usage of the augmenting paths discussed in 2.2. In the former case all the previously discussed augmenting functions (see Chap. 2) were included, while in the latter case none of these methods were

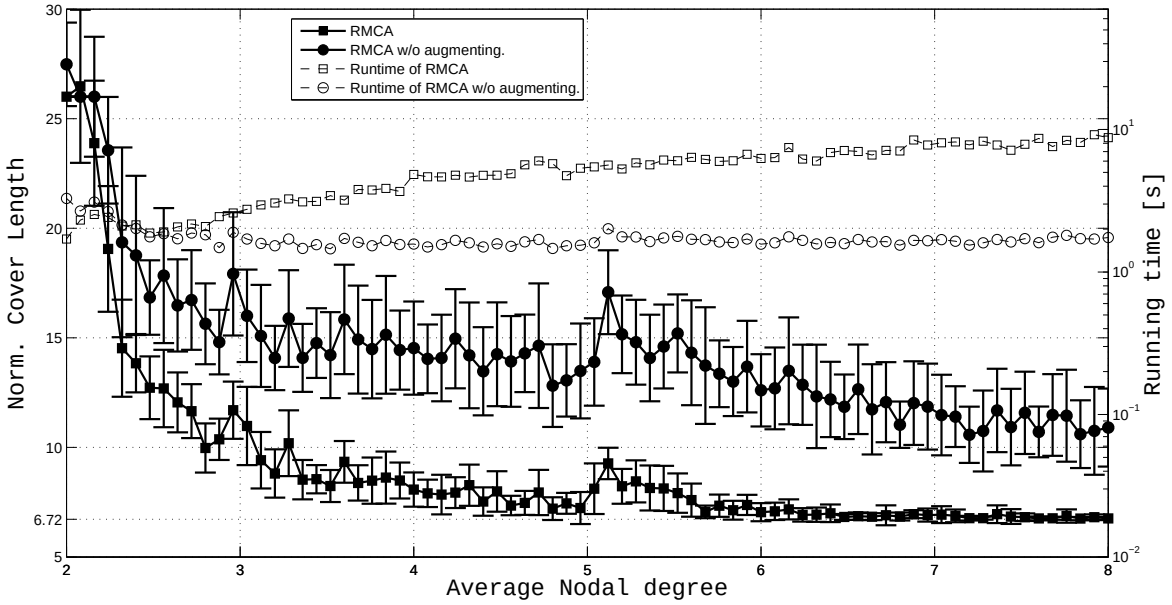


Figure 3.5: *Results of 380 randomly generated planar graphs with 50 nodes.*

utilized for increasing the matching cardinality. The results are shown on Fig. 3.5, where the corresponding values for each nodal degree are represented with error bars, showing the mean $||\mathcal{T}||_V$ value of the 25 results, and their standard deviation. As the input graph contains 50 nodes, the minimum attainable value of $||\mathcal{T}||_V$ is $\lceil \log_2 n \rceil + \kappa = 6.72$ (see Eq. (2.1)), indicated with an additional tick on the y-axis, and a dotted line. Note, that RMCA with augmenting paths not only approaches the optimum much quicker, but it also provides significantly smaller variance. One of the main importances of this figure, however, is that RMCA performs well on graphs with rather low average nodal degree. As mentioned before, we are mainly interested in backbone optical networks with nodal degree around 3-4.

The figure also shows the mean running time values for a graph with a given nodal degree (as the average of the corresponding 25 execution time values). Although, while the performance of RMCA in terms of $||\mathcal{T}||_V$ values in this case is very similar to the behaviour seen at Section 3.1 (compared to the 50-node curve on Figure 3.2), the execution time changes with the nodal degree in a different manner. In particular, in case of planar graphs the average execution time went from one second to almost 10 seconds with the growth of nodal degree (note, that 10 seconds is still regarded as a low execution time). The main reason for this behavior is obviously the difference of the topologies, particularly, the diameter of the two types of graphs. The distance $d_G(u, v)$ in a graph G is the minimal number of edges of a path connecting u to v (i.e. their degree of separation). The diameter of a graph is the supremum of these distances over all the pairs of nodes u and v:

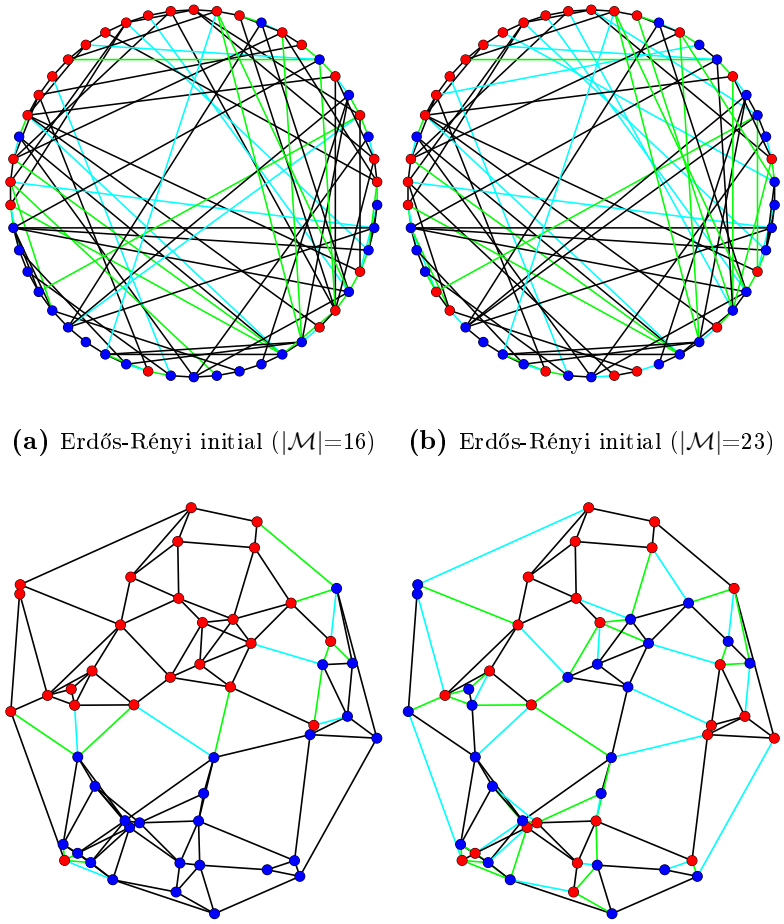
$$D(G) = \sup_{\forall(u,v)} d_G(u, v) \tag{3.1}$$

The randomness during Erdős-Rényi graphs' creation ensures both "long" and "short" jumps in the network, so that the diameter is expected to be very low in contrast with a planar topology, where additional edges are more likely to be established in between two nodes that are already close to each other, thus their distance from the furthest node in the network decreases much slower. This behaviour of the color classes' subgraphs in planar graphs implies, that most of the edges will be *intra*-class edges after the initial coloring step of the algorithm, that result in a relatively low number of inter-class edges, thus a low initial matching cardinality. In case of Erdős-Rényi graphs the initial number of *inter*-class edges is expected to be larger, as its low diameter suggests, hence, the initial matching cardinality is also expected to be relatively large. Fig. 3.6 shows a comparison of an Erdős-Rényi and a planar graph's initial, and augmented matching (pale blue edges are the ones included in the current matching, green ones are the remaining inter-class edges). Note, that the initial matching cardinality of the Erdős-Rényi graph is almost 3 times more, than the planar's. Despite this significant difference, we have observed very similar curves for their performances in terms of $||\mathcal{T}||_V$. The main reason for this is the contribution of the newly defined MAPs. With the help of these matching-cardinality increasing mechanisms, an initially low matching-size can easily approach the maximum possible number of paired edges (see the augmented matchings on 3.6)

The drawback obviously is the time consumption of these augmenting steps. This is behaviour that we can examine on Fig. 3.5. With the growth of the average nodal degree, the number of intra-class edges increases as well, and as the defined MAPs utilize these edges as well, the number of augmenting steps will grow, that leads to the experienced execution time performance.

3.3 Real topologies

The proposed RMCA algorithm has also been tested on 10 well-known optical networks, taken from [8]. The graphs are included on Fig. 3.7. As a comparison we used the Greedy Link Swapping heuristic for node failures GLS_{node} in [11]. The objective in the GLS_{node} heuristic is to minimize the total number of m-trails, while in this scenario the objective is to minimize the average number of m-trails seen at each node of the network. Note that the RMCA algorithm finds solutions with smaller m-trails than GLS_{node}, which results smaller average number of m-trails traversing the nodes even if the total number of m-trails is larger. The algorithm was executed on each topology 100 times on a commodity laptop with 1.8 GHz core i5 CPU and 4 GB RAM. The results are summarized in Table 3.1 along with the running time for a single execution, the corresponding results of GLS_{node}, and the theoretical lower bound of $||\mathcal{T}||_V$, $\lceil \log_2 n \rceil$. Note, that most of the cases not only the minimum, but also the mean values of $||\mathcal{T}||_V$ provided by RMCA are significantly better than the results of GLS_{node} algorithm. Moreover, the running time of a single execution of RMCA is most of the times at least an order of magnitude smaller then the GLS_{node} method, while the best $||\mathcal{T}||_V$ reached by RMCA is on average 24% better than the values obtained by GLS_{node} proposed in [11]. An exceptionally convincing improvement of 39.4%



(a) Erdős-Rényi initial ($|\mathcal{M}|$=16) **(b)** Erdős-Rényi initial ($|\mathcal{M}|$=23)

(c) Planar initial ($|\mathcal{M}|$=6) **(d)** Planar augmented($|\mathcal{M}|$=22)

Figure 3.6: *Comparison of initial color classes of an Erdős-Rényi graph and a planar graph, both with n=50 and nodal degree 4.8*

is achieved on the 22-node European network, possibly because of its fairly large nodal degree.

To further investigate the running time behaviour of the two algorithms, their time-evolution characteristics were plotted through the 100 executions. Fig. 3.9 shows the best solution found versus the running time for the 10 optical networks.[1] In these networks RMCA turned out to find solution with the same quality 1000-10000 times faster than GLS_{node}. The beginning of the lines represent the first valid solutions of the corresponding algorithms, and the improvements over time are depicted as stair functions. Note, that RMCA provides feasible solutions earlier, and their quality is also significantly better, than the ones given by GLS_{node}. Despite the continously improving results, 9 cases out of 10, GLS_{node} could not reach even the initial solution of RMCA in 100 executions on the shown examples.

As a last simulation scenario, I have investigated the possible effects of the lack of certain components in RMCA. Five different RMCA-versions were used during the simulation:

[1]Note, that the lines' last point represents the last improvement, not the end of the 100 executions

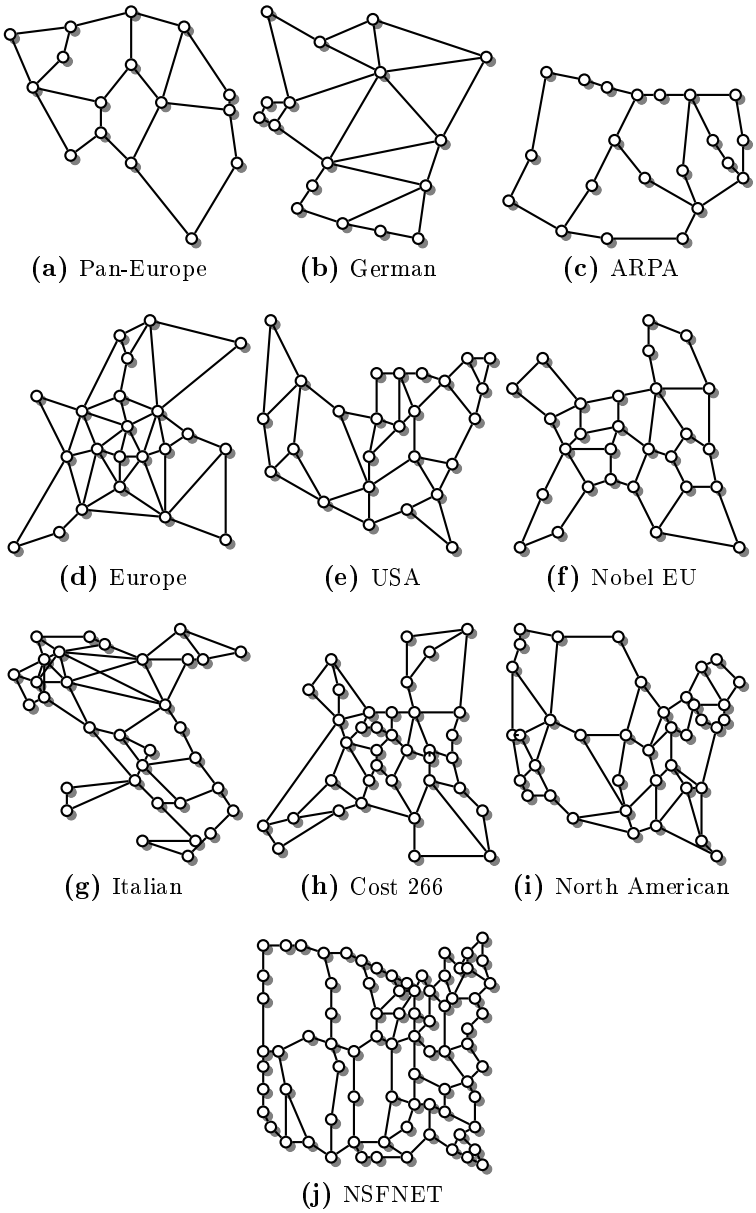


Figure 3.7: *The 10 backbone network topologies.*

- RMCA including all the previously discussed augmenting mechanisms
- RMCA without the usage of Augmenting path and MAPs (2.2.2, 2.2.3, 2.2.4)
- RMCA without unnecessary m-trail elimination (2.2.7)
- RMCA without cut vertex elimination mechanism (2.2.6)
- RMCA without all the above mechanisms

Each type of RMCA has been executed 100 times on the 10 optical networks. The results are summarized on a barplot on Fig. 3.8. Besides the best results of the different RMCA algorithms, the lowest $||\mathcal{T}||_V$ values provided by GLS_{node} (see in 3.1) are included on the barplot as the 6th bars.

Table 3.1: *Results on some well-known optical networks*

Network	Graph			Results of GLS_{node}		Results of RMCA			Improvement [%]
	n	Nodal degree	$\lceil \log_2 n \rceil$	Best $\|\|\mathcal{T}\|\|_V$	Runtime [s]	Mean $\|\|\mathcal{T}\|\|_V$	Best $\|\|\mathcal{T}\|\|_V$	Runtime [s]	
Pan-European	16	2.75	4	8.31	0.38	6.853	6.187	0.13	25.6
German	17	3.06	5	8.71	0.9	8.304	6.765	0.15	22.3
ARPA	21	2.38	5	10.33	0.83	8.623	7.380	0.19	28.6
European	22	4.09	5	10.5	2.24	7.779	6.364	0.24	39.4
USA	26	3.23	5	11.38	3.31	10.77	9.384	0.34	17.5
Nobel EU	28	2.93	5	12.42	3.04	10.675	8.643	0.38	30.4
Italian	33	3.4	6	14.09	10.52	12.375	9.606	0.66	31.9
Cost 266	37	3.08	6	11.75	7.56	11.308	8.540	0.82	27.3
North American	39	3.13	6	12.43	10.09	12.812	10.513	1.18	15.4
NFSNET	79	2.73	7	17.68	68.7	20.868	16.582	7.92	6.2

Surprisingly, RMCA without the usage of any augmenting methods performs better than GLS_{node} on the first 7 graphs. Although, as the graphs sizes are increasing, the need for improving procedures is rising as well. This few example suggests, that MAPs, and cut vertex removals contributes the most for the initial matching-contraction on large graphs, nevertheless, the lack of trail-reduction step has a remarkable effect as well.

In conclusion, the novel Recursive Matching Contraction Algorithm performs convincingly on a wide range of different topologies. It works especially well on graphs with large average nodal degree, however, due to the previosly introduced matching cardinality increasing augmenting paths, it manages to solve sparse networks quite efficiently as well.

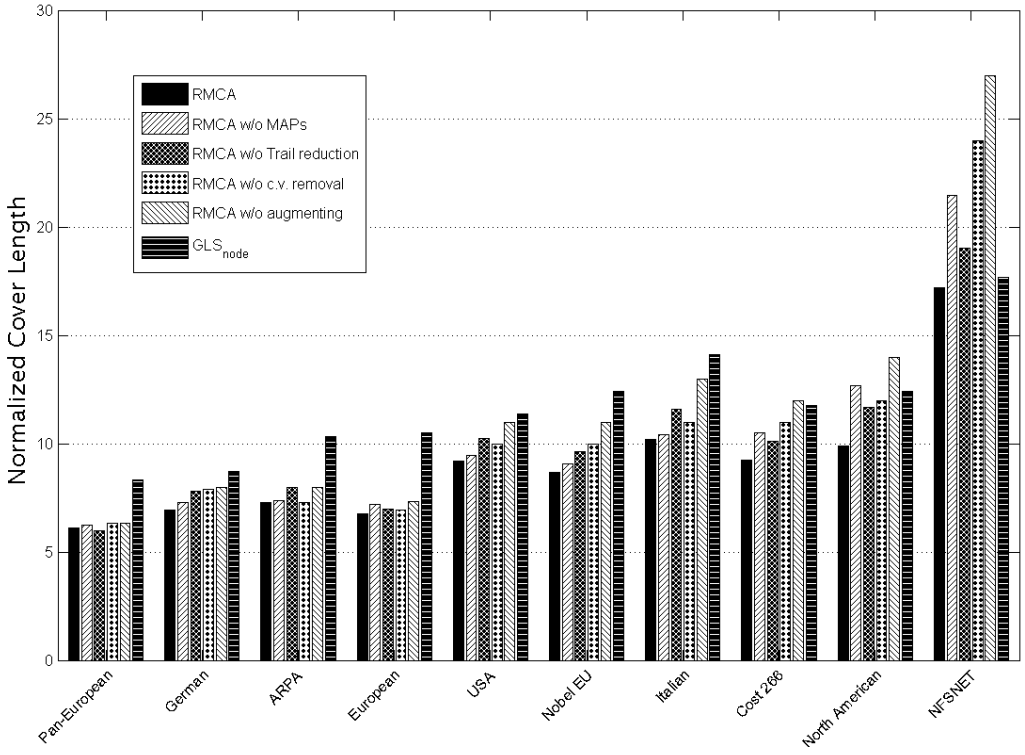
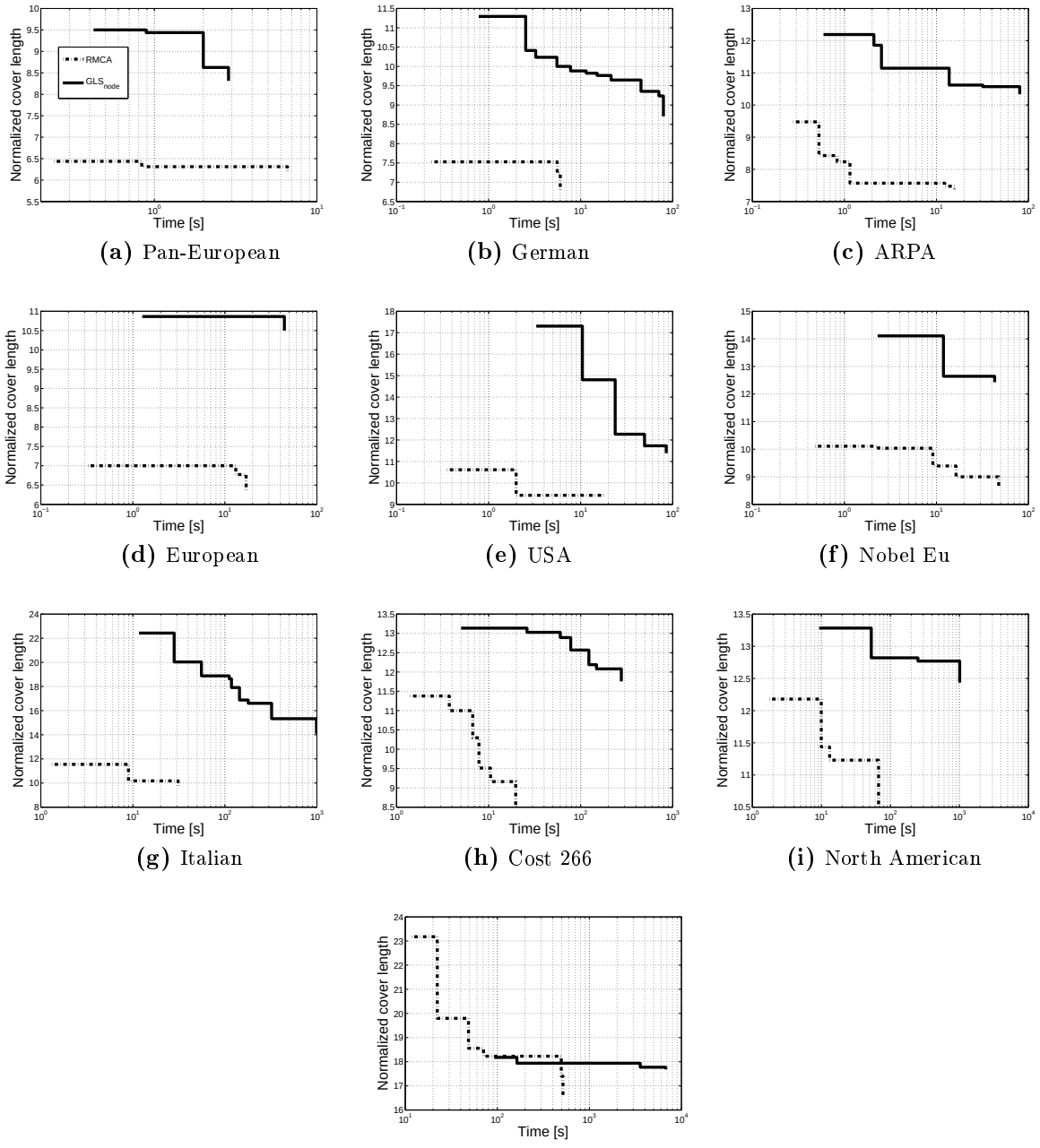


Figure 3.8: *Comparison of different variations of RMCA, and GLS_{node}.*



(a) Pan-European **(b)** German **(c)** ARPA

(d) European **(e)** USA **(f)** Nobel Eu

(g) Italian **(h)** Cost 266 **(i)** North American

(j) NFSNET

Figure 3.9: *The best found solutions versus the running time after launching the algorithms 100 times on the 10 networks. The results of RMCA is drawn with dash-dot line, while the GLS_{node} with solid line.*

Chapter 4

Conclusion

The issue of network-wide unambigious node failure localization (NL-UNFL) through supervisory lightpaths (monitoring trails) was presented in this study. The scheme of monitoring trails is considered to be a simple, yet effective tool for fast failure localization in all-optical networks, where certain nodes, called monitors, can inspect the on-off statuses of these multihop supervisory paths. Our goal in the NL-UNFL problem is to ensure, that every node v can unambiguously identify any single node failure in the network solely by inspecting its locally available monitoring trails via signal tapping.

I have presented a novel heuristic algorithm, called RMCA, that uses recursive matching-contractions to assign unique binary codes to the nodes. The main advantage of the proposed method is that it recursively contracts certain nodes in the input graph, so that it rapidly provides feasible solutions even for large inputs. Minor modification steps, such as unnecessary m-trail removal or articulation point elimination were introduced in orther to further improve RMCA's performance.

Simulations were conducted both on a large number of randomly generated graphs and on some well-known optical networks in order to evaluate the proposed algorithm's computation time and its solution quality in terms of normalized cover length, that is the average number of m-trails seen at each node in the network. Simulations on Erdős-Rényi and planar graphs revealed that the diameter of the input graph significantly influences the proposed algorithm's execution time. Our experiments on the well-known networks showed that the RMCA convincingly outperforms the previously known best algorithm (GLS_{node}) both in terms of normalized cover length and running time. On real topologies, RMCA has found solutions with the same quality as GLS_{node} 3-4 order of magnitude faster.

The results of my research are summarized in a paper, submitted to the ICC 2015 conference [5].

List of Figures

List of Tables

Bibliography

[1] LEMON: A C++ Library for Efficient Modeling and Optimization in Networks. http://lemon.cs.elte.hu.

[2] S.S. Ahuja, S. Ramasubramanian, and M. Krunz. Single link failure detection in all-optical networks using monitoring cycles and paths. *IEEE/ACM Trans. Networking*, 17(4):1080–1093, 2009.

[3] C. Assi, Y. Ye, A. Shami, S. Dixit, and M. Ali. A hybrid distributed fault-management protocol for combating single-fiber failures in mesh based DWDM optical networks. In *Proc. IEEE GLOBECOM*, pages 2676–2680, 2002.

[4] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–456, 1965.

[5] L. Gyimothi and J. Tapolcai. A heuristic algorithm for network-wide local unambiguous node failure localization. In *IEEE ICC*, 2015.

[6] W. He, Pin-Han Ho, Bin Wu, and J. Tapolcai. On identifying SRLG failures in all-optical networks. *Elservier Journal on Optical Switching and Networking (OSN)*, 10(1):77 – 88, jan 2013.

[7] C.S. Li, R. Ramaswami, I.B.M.T.J.W.R. Center, and Y. Heights. Automatic fault detection, isolation, and recovery in transparentall-optical networks. *IEEE/OSA J. Lightwave Technol.*, 15(10):1784–1793, 1997.

[8] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. SNDlib 1.0–Survivable Network Design Library. In *Proc. Int. Network Optimization Conference (INOC)*, April 2007.

[9] J. Tapolcai, Pin-Han Ho, P. Babarczi, and L. Rónyai. *Internet Optical Infrastructure - Issues on Monitoring and Failure Restoration*. Springer, 2014.

[10] J. Tapolcai, Pin-Han Ho, L. Rónyai, and Bin Wu. Network-wide local unambiguous failure localization (NWL-UFL) via monitoring trails. *IEEE/ACM Transactions on Networking*, 2012.

[11] J. Tapolcai, L. Rónyai, É. Hosszu, Pin-Han Ho, and S. Subramaniam. Signaling free localization of node failures in all-optical networks. In *Proc. IEEE INFOCOM*, pages 1860–1868, Toronto, Canada, May 2014.

[12] Y. Wen, V.W.S. Chan, and L. Zheng. Efficient fault-diagnosis algorithms for all-optical WDM networks with probabilistic link failures. *IEEE/OSA J. Lightwave Technol.*, 23:3358–3371, 2005.

[13] Bin Wu, Pin-Han Ho, J. Tapolcai, and X. Jiang. A novel framework of fast and unambiguous link failure localization via monitoring trails. In *IEEE INFOCOM WIP*, pages 1–5, San Diego, 2010.

[14] H. Zeng, C. Huang, and A. Vukovic. A Novel Fault Detection and Localization Scheme for Mesh All-optical Networks Based on Monitoring-cycles. *Photonic Network Communications*, 11(3):277–286, 2006.