A Heuristic Algorithm for Network-Wide Local Unambiguous Node Failure Localization

László Gyimóthi, János Tapolcai
MTA-BME Future Internet Research Group, Dept. of Telecommunications and Media Informatics,
Budapest University of Technology and Economics, {tapolcai, gyimothi}@tmit.bme.hu

***Abstract*—This paper deals with fast node failure localization in optical networks with monitoring trails (m-trails). It is based on Network-wide local unambiguous failure localization, which enables every node to autonomously localize any single node failure in the network in a distributed and all-optical manner by inspecting the m-trails traversing through the node. A new and innovative heuristic algorithm is presented which is based on recursion and constrained matching algorithms in general graphs. Extensive simulation is conducted to examine the proposed heuristic in terms of the required cover length to achieve NL-UFL. In our experiments the new heuristic can reduce the computation time by 1000-10000 times compared to prior art.**

I. Introduction

The problem investigated in this paper is related to optical layer restoration, which allows full reconfiguration of the optical network to survive failures. Although restoration can achieve the theoretically minimal protection capacity, implementing restoration requires an intensive synchronization in the restoration process within 50 msec after the failure, which leads to long recovery time, and makes the mechanism not popular in practice. For example implementing stub release[1] in real time requires an exact knowledge of the failed elements (links and nodes) in the network for each node. To fill this gap, in the paper we focus on fast failure localization techniques for single node failures.

The traditional approaches use control plane packets to exchange the status information of the links and nodes after failures. When multiple failures are considered the general approach in network planning is defining the set of failure states the network can survive, called *shared risk link groups* (SRLG)[2]. This eventually results that after failure the ID of the failed SRLG should be disseminated. An alternative idea was to follow the compressed sensing approach, where a set of supervisory lightpaths are allocated in the network. Each supervisory lightpath, a.k.a. *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 [1]–[8]. A detailed overview of fast failure localization with the m-trails can be found in the book [2].

L-UFL was first investigated in [7] 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. These MLs can reconfigure the network after the failure in a centralized manner. To reduce the recovery time [1] 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. It requires every node to be L-UFL, called Network-wide L-UFL (NL-UFL), which allows a significantly shorter recovery time as the network reconfiguration after the failure is carried out without any control packet, just using the local failure information available at each node. The study [1] focused on the problem of allocating m-trails capable for NL-UFL for any single link failure. Link failures are more common than node failures, however the failure of a node has a larger influence on the network configuration, because all the connections terminating in the failed node can be released. Therefore localizing node failures is also important for optical layer restoration. However, for link failures the efficient heuristics allocate only m-trails that traverse every node in the network, which is not capable of localizing node failures, as a failure of the node would interrupt each m-trail in the network. Localizing node failures in NL-UFL is a rather challenging problem and the only paper reported was [8]. It presents a heuristic algorithm that generates a random initial solution that is possibly invalid. 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 a slow process[3]. This function is launched hundreds of times, which end ups a rather time consuming job.

In this paper we follow the NL-UFL model of [8] and the goal is to localize single node failures only. Note that the m-trails can localize a single failure, thus to be able to identify a next failure after the first failure the m-trails should be

Research partially supported by the Hungarian Scientific Research Fund (grant No. OTKA 108947).

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

[2]The rest of failure events are unlikely and thus ignored at the planning stage.

[3]at least $|SRLG| \cdot |V|$ steps

reconfigured. Therefore the computation time of the NL-UFL design algorithm is an important factor in the applicability of the approach. The main contribution of the paper is to present a new heuristic algorithm for the problem which is four order faster then the prior art. We take a novel approach which ensures the validity of the solution by limiting the problem space to specific m-trail constructions.

The paper is organized as follows. Section II defines the m-trail problem. Section III introduces the proposed heuristic algorithm on general graphs and Section IV shows simulation results which verify the proposed heuristic algorithm. Section V concludes the paper.

II. 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 is to establish a set of m-trails, denoted by $\mathcal{T} = \{T_1, \ldots, T_b\}$ 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 $\mathcal{T}^j = \forall T_i | v_j \in T_i$ - the subset of $\mathcal{T}$ containing the m-trails passing through v_j. In this paper we focus on single node failures only, and assume the links are perfect. Thus, compared to the prior art, the m-trails are defined as a set of nodes, instead of a set of links. After the m-trails are computed links are selected to form a closed trail in the network through the nodes. Therefore, we need to ensure each m-trail $T_i \subseteq V$ is composed of a set of nodes defining a connected component of 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): The failure of every node $v_j \in V$ results a unique alarm code at every node $v_k \in V$. Formally, $\forall v_j, v_k, v_l \in V$ there is an m-trail $T_i \in \mathcal{T}$ such that $v_l \in T_i$, and either $v_j \in T_i$ and $v_k \notin T_i$, or $v_j \notin T_i$ and $v_k \in T_i$.
- (R2): m-trail $T_i \in \mathcal{T}$ is a set of nodes forming a connected subgraph of G, for $i = 1, \ldots, b$.

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

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

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.

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, see also Fig. 1 as an example. 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. 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.

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)

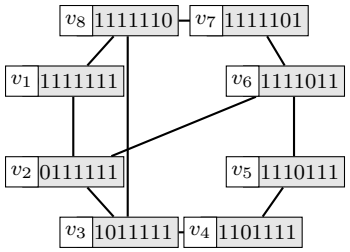
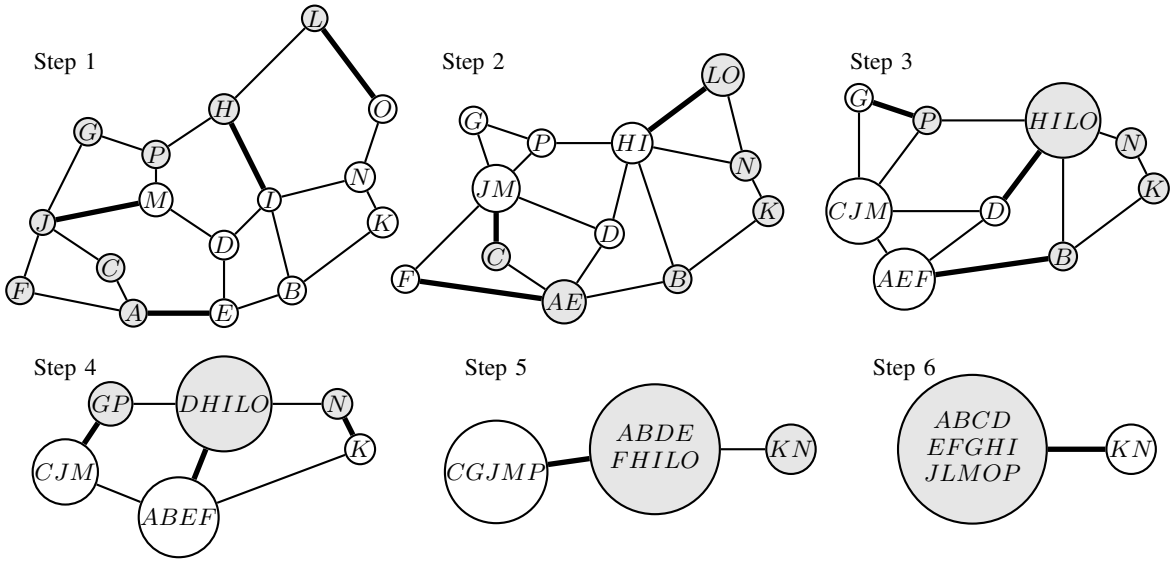


Fig. 1. Implementing NL-UNFL in a simple way on a 2-connected graph with 8 nodes

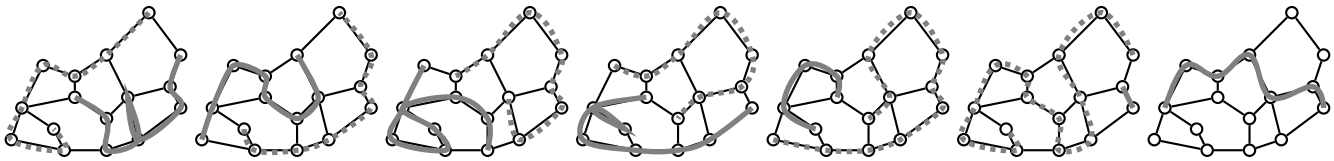
III. Recursive Matching-Contraction Algorithm (RMCA)

Inspired by the m-trail construction for star graphs given in [1] we define a general construction for NL-UNFL. 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 subgraphs composed by the 1s and the 0s are connected. See also Fig. 2 as an example. In this case NL-UNFL is achievable with $2k + 1$ m-trails. It is because 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. This procedure guarantees, that each node is traversed by exactly k trails, and can distinguish all the possible single node failures. 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 unambiguous node failure localization at every node.

Assigning unique binary codes in a way mentioned above is not trivial because we need to ensure that at every bit position, both the subgraphs composed by the 1s and the 0s are connected and also to keep the value of k as low as possible. A possible solution could be the following:



(a) The graph and the maximal matching with bold edges and the two connected color class after merging in each step. 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, and each step defines two new m-trails.



(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. This last trail will increase $||\mathcal{T}||_V$ by $\frac{7}{16}$, as it traverses 7 vertices in the 16-node network.

Fig. 2. An illustrative example of the algorithm on 16-node European reference network. Fig. (a) shows the sequence of merging steps, where each merging step defines two new m-trails representing the two disjoint color classes, colored with white and grey. The importance of the connectivity of both color classes is shown by Fig. (b).

- Divide the vertices of the input graph $G = (V, E)$ into two disjoint color classes in such a way, that both classes compose a connected subgraph of G. Find a maximum matching $\mathcal{M}$ among them. Vertices in the same classes are assigned the same binary value at the current bit position. In other words two m-trails are defined, each containing the nodes of a color class.
- Contract $\forall e \in \mathcal{M}$ resulting a new graph $G' = (V', E')$.
- Save the two new m-trails and 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 distinguished further on from each other. Thus, their contraction will not affect 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 are assigned a unique binary code. Moreover, the subgraphs of both the 1s and the 0s are connected at each bit position.

Obviously, the hardest part of the algorithm described above is finding two disjoint connected subgraphs with maximum matching among the nodes of the two subgraphs. Edmonds' algorithm, described in [9], finds a maximum matching in a general graph in polynomial time, however, it does not take into account the connectivity constraint of the nodes in each color classes, therefore it cannot be used as is for our case. In the next section we propose a heuristic inspired by Edmonds' algorithm. 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 proposed recursive matching-contraction algorithm (RMCA) can reach a $||\mathcal{T}||_V$ value of $\lceil \log_2 n \rceil + \kappa$, where $\kappa = \frac{2 \cdot (n - 2^{\lceil \log_2 n \rceil - 1})}{n}$, $0 < \kappa \leq 1$. In order to achieve this minimum 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 $n - 2^{\lceil \log_2 n \rceil - 1}$ complement codes will be obtained, as there are $2^{\lceil \log_2 n \rceil - 1}$ 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. 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.

A. Heuristic to Find Maximum Matching with Nodes in Each Color Class Being Connected

The solution given by Edmonds' algorithm contains the maximal matching in the graph obtaining two color classes. We call the edges with the same color and both ends as intra-class edges, otherwise it is an inter-class edge. In the subproblem we are facing we also need to ensure that the nodes in each color class can be connected with a trail on intra-class edges. A promising idea to use a local search heuristic, that adapts the augmenting paths technique of the matching algorithms, but ensures the connectivity of the color classes at the same time. In other words it tries to maximize the cardinality of an initial matching by means of certain augmenting paths. 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 in different color classes, and 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 $\mathcal{M}$. 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 is implemented in the heuristic.

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. If the graph is connected (as we always assume), the walks will eventually cover all the nodes in the network classifying them into $\mathcal{R}$ and $\mathcal{B}$ classes. 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 augmenting paths. 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 (MAP).

1) 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 $r_2 \in \mathcal{R}$ 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 $\mathcal{M}$. 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 cease the connectivity of R. Note, that not all the edges are included on Fig. 3, only the ones that are directly utilized during the augmenting method and the ones included in $\mathcal{M}$ (bold edges).

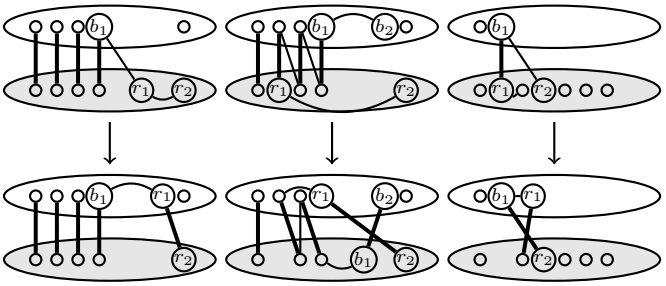


Fig. 3. An example of the 3 different MAPs

2) 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 from a pair vertex $r_1 \in \mathcal{R}$ to $b_1 \in \mathcal{B}$, and both r_1 and b_1 are connected to an unpaired vertex in their own color class, then an augmenting path is found (see Fig. 3). Note, however, that neither r_1 nor b_1 can be cut vertices in their class, since the class-connectivity constraints have to be satisfied.

3) MAP type 3 + Spreading: 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 resulting matching will probably have a suboptimal cardinality. A possible way to overcome this issue is to somehow force the smaller class spread into the larger one. Fig. 3 shows one step of spreading. If the larger color class contains a paired vertex r_1 that is not a cut vertex, and is connected to a not paired vertex within its class, then it can be moved to the smaller class. Spreading does not increase the cardinality of $\mathcal{M}$, it only balances the number of the vertices in the two color classess. 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 $\mathcal{M}$. This method can be the 3rd type of augmenting paths.

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

B. Articulation Point Removal

The proposed RMCA performs very well on graphs with large average nodal degree, nevertheless, it has some limitations on real 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.

The removal of the cut vertex, and connecting its neighbour vertices with each other would solve this issue. In order to justify this step, four additional monitoring trails have to be added to our set: two for the two components containing the cut vertex, and two more monitoring trails for the two components without the cut vertex. The unambiguous localization of the cut vertex in the whole network is ensured this way. The drawback for the newly added edges after the 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.

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: $G \setminus \{v\}$.

C. 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 III-B. 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 $\mathcal{T}$.

IV. Simulation results

The proposed algorithm was tested on a large number of different input topologies including graphs representing real optical networks. As a comparison we used the Greedy Link Swapping heuristic for node failures GLS$_{node}$ in [8]. 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 might be larger.

First, we generated a large number of planar graphs with large diameter. We used the a random graph generator for creating optical network-like topologies [2].We have increased the average nodal degree of a 50-node graph by 0.08 steps from 2.0 to 10.0, at every step 5 different graphs were generated, resulting in a total number of 500 graphs. 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 III-A. In the former case all the previously mentioned augmenting paths, and cut vertex removals were included. In the latter one, however, none of these methods were utilized for increasing the matching cardinality, the matching achieved after the initial coloring was used for all the cases. The results are shown on Fig. 4 where the 95% confidence intervals are drawn with error bars, and also the minimum possible $||\mathcal{T}||_V$ value, provided by the heuristic algorithm. 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 III), indicated with a dash-dot line on Fig. 4 as the theoretical minimum. Note, that RMCA with augmenting paths not only approaches the optimum much quicker, but it also provides significantly smaller variance. The main importance 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 an average nodal degree value of 3-4.

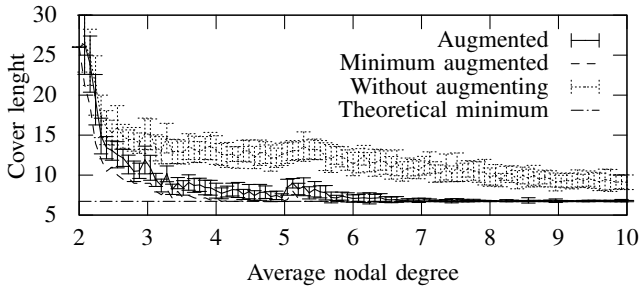


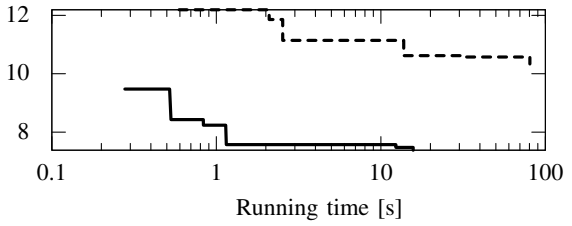
Fig. 4. Results of 500 randomly generated planar graphs with 50 nodes.

The proposed algorithm has also been tested on 10 well-known optical networks, taken from [11]. 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 I 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 in 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. Furthermore, the running time of a single execution of RMCA is most of the times almost an order of magnitude smaller then the GLS$_{node}$ method's, while the best $||\mathcal{T}||_V$ reached by RMCA is on average 23% better than the values obtained by GLS$_{node}$.

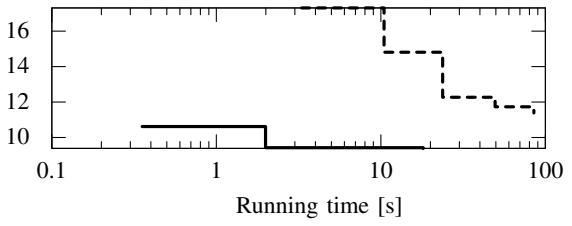
To compare the running time of the two approaches we launched both algorithms 100 times for the networks of Table I. Fig. 5 shows the best solution found versus the running time for 4 selected networks. In these networks RMCA turned out to find solution with the same quality 1000-10000 times faster than GLS$_{node}$. For example for the North American network GLS$_{node}$ found a solution with $||\mathcal{T}||_V = 12.44$ after running 1009 seconds, while RMCA for the first run found a better

TABLE I
RESULTS ON SOME WELL-KNOWN OPTICAL NETWORKS

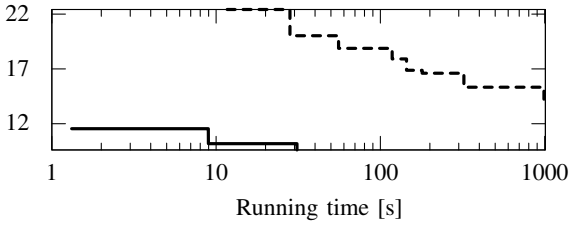
Network	Graph			Results of GLS_{node}		Results of RMCA			Improvement [%]
	n	Nodal degree	$\lceil \log_2 n \rceil$	Best $\|\|\mathcal{T}\|\|_V$	Runtime[s]	Mean $\|\|\mathcal{T}\|\|_V$	Best $\|\|\mathcal{T}\|\|_V$	Runtime[s]	
Pan-European	16	2.75	4	8.31	0.38	6.853	6.187	0.13	25.6
German	17	3.06	5	8.71	0.9	8.304	6.647	0.15	23.7
ARPA	21	2.38	5	10.33	0.83	8.623	7.571	0.19	26.7
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	9.115	0.34	19.9
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	10.606	0.66	24.7
Cost 266	37	3.08	6	11.75	7.56	11.308	9.081	0.82	22.7
North American	39	3.13	6	12.43	10.09	12.812	10.333	1.18	16.9
NFSNET	79	2.73	7	17.68	68.7	20.868	17.379	7.92	1.7



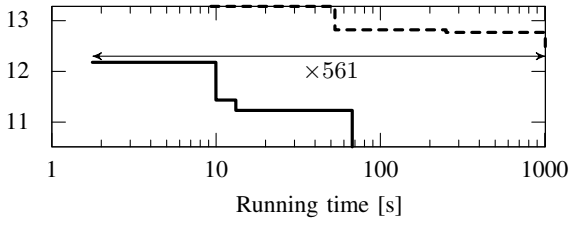
(a) ARPA



(b) USA



(c) Italian



(d) North American

Fig. 5. The best solution versus the running time after launching the algorithms 100 times. The results of RMCA is drawn with solid line, while the GLS_{node} with dashed line.

solution with $||\mathcal{T}||_V = 12.18$ after 1.76 seconds, which means RMCA is at least times 561 faster than GLS_{node} obtaining the same solution quality.

V. CONCLUSION

In the paper we investigated the network-wide unambiguous failure localization problem for single node failures, using supervisory lightpaths (monitoring trails). We presented a novel heuristic method, that uses recursive matching-contractions to assign binary codes to the nodes, and rapidly provides feasible solutions. Simulations were conducted both on 2-connected randomly generated planar graphs and on some well-known optical networks in order to evaluate the proposed algorithm's computation time and performance in terms of average number of m-trails seen at each node in the network. Results show, that RMCA find the solution with the same quality 3-4 order of magnitude faster than the previously known best algorithm.

REFERENCES

[1] J. Tapolcai, P.-H. Ho, L. Rónyai, and B. Wu, "Network-wide local unambiguous failure localization (NWL-UFL) via monitoring trails," *IEEE/ACM Transactions on Networking*, 2012.

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

[3] 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*, vol. 11, no. 3, pp. 277–286, 2006.

[4] C. Li, R. Ramaswami, I. Center, and Y. Heights, "Automatic fault detection, isolation, and recovery in transparentall-optical networks," *IEEE/OSA J. Lightwave Technol.*, vol. 15, no. 10, pp. 1784–1793, 1997.

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

[6] 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*, 2002, pp. 2676–2680.

[7] S. Ahuja, S. Ramasubramanian, and M. Krunz, "Single link failure detection in all-optical networks using monitoring cycles and paths," *IEEE/ACM Trans. Networking*, vol. 17, no. 4, pp. 1080–1093, 2009.

[8] J. Tapolcai, L. Rónyai, E. Hosszu, P.-H. Ho, and S. Subramaniam, "Signaling free localization of node failures in all-optical networks," in *Proc. IEEE INFOCOM*, Toronto, Canada, May 2014, pp. 1860–1868.

[9] J. Edmonds, "Maximum matching and a polyhedron with 0, 1 vertices," *J. of Res. the Nat. Bureau of Standards*, vol. 69 B, pp. 125–130, 1965.

[10] 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.