CMST 25(1) 7–15 (2019) DOI:10.12921/cmst.2018.0000038

Alternative route search algorithm for robust and balanced traffic in telecommunication network

M. Wiktor

ADVA Optical Networking
E-mail: miwiktor@o2.pl

Received: 10 July 2018; revised: 29 January 2019; accepted: 04 February 2019; published online: 31 March 2019

Abstract: This paper discusses routing policy in optical transport networks. Dijkstra's shortest path algorithm is compared to a new path computation technique based on heuristics originated from human behavior combined with spectral graph embedding. The two-step procedure allows one to separate the computationally expensive and computationally cheap parts for efficient implementation in network infrastructure.

Key words: optical transport networks, path computation, spectral graph embedding

I. INTRODUCTION

Optical transport networks are the core of modern telecommunication infrastructure. Data transferred over the network must cross a number of separate optical links and intermediate nodes, where the data are routed in a desired direction. Compared to computer networks, where each portion of data is transmitted and routed independently, in core transport networks there are predefined paths called tunnels. When a tunnel is put into a service, the data sent to a transmitter are sent to a receiver at the second end of the tunnel. Crossing the network is transparent for the data and the user. Creating a tunnel is a demanding and time-consuming task, involving path calculation and setting appropriate devices to route the optical signal in a desired way and equalizing power levels. In general, routing policy in an optical transport network must meet numerous, sometimes mutually exclusive requirements. The most obvious demand is to choose the shortest possible path between network nodes. This intuitively obvious requirement, however, is not the most important objective. When planning a route, many other factors must be considered, like the existence of a disjoint protection path, global power consumption, or power levels in optical link. The physical distance is rarely the point; far more important is a parameter called logical distance, which is measured in the number of routers and regenerators between ingress and egress nodes [2, 3]. In terms of graph terminology, the distance between nodes can be defined as the number of intermediate nodes within the route.

For the purpose of route calculation, Dijkstra's shortest path algorithm is extensively used. It guarantees that the route will be found if only such a route exists. It is also guaranteed that the route will be the shortest possible one. By defining appropriate weights of links, represented as graph edges, one may tune the path in order to satisfy different, previously mentioned, objectives.

II. MOTIVATION AND BACKGROUND

The method of path computing presented in this paper is dedicated to a specific model of a network. A number of objectives can be assumed for an optical transport network; however, according to typical patterns of use [8] the following assumptions can be made:

II. 1. Equal link weight

Beside very specific applications, like low latency networks [11], the time needed for sending the signal from the source to the destination point can be assumed to be independent of the physical length of the fiber. This assumption can seem strange. However, even for fully optically switched network, time required for light propagation in photonic crystals and prisms, encoding and decoding signal at ingress and egress nodes lasts longer than its propagation along optical fibres. For this reason we have decided to neglect the physical length of the link and expose the number of chops as a route cost [1].

II. 2. Centralized and distributed path calculation

There are two scenarios when an operator wants to create a new optical tunnel. The first scenario, for centralized resource allocation, assumes the request is sent to a master node of a network where the path is calculated and corresponding signaling is spread over the network. Centralized management, when a master node knows the network state is very efficient in terms of load balancing, but centralized network management is less robust due to possible link malfunction.

In case of distributed resource allocation, although nodes are aware of the network resources, they are only partially aware of the network state, specifically, which of the resources are in use. In other words, two separate nodes can see a resource as free and request for its use simultaneously. On the other hand, such a network is much more stable and robust in case of node or link failure. For our considerations, the distributed model has been chosen. This implies that the path computation algorithm is not aware of the state of the network, especially how the traffic is provisioned when the path is calculated. On the one hand, it deteriorates global traffic optimization and, on the other hand, this makes the algorithm fully deterministic.

II. 3. Existing approaches and industrial implementations

Despite of the network model, the problem of searching node disjoint or path disjoint paths is critical in network applications. According to the theoretical background presented in [9], both formulations are NP-complete, which implies using heuristic algorithms to solve the problem. Interesting results and wider context was presented in [10]. Good results in terms of both the cost of a generated route and solution time are presented herein. The core of the algorithm is multiple path calculation with Dijkstra shortest path algorithm and simultaneous reduction of the graph.

The telecommunication applications, however, do not follow research. The cost of keeping backward compatibility and protocol support is restriction to Dijkstra algorithm. However, even for well-defined Dijkstra method, the choice between paths having the same cost is a task the requires additional resources. For telecommunication applications the routing algorithm must be computationally cheap and easy to implement.

II. 4. Proposed approach

According to the discussion presented in the previous section, we introduce a routing algorithm for network managed in a distributed manner, and all optical links having the same weight, despite of their length. The direct consequence of this approach is that the algorithm is not aware of the network state, especially links load. In this terms the algorithm is deterministic, contrary to nondeterministic approaches, where the links weight can be changed as the link load grows.

Since all states of the art algorithms [10] are computationally expensive, we propose the following workaround: the proposed heuristics is split into two phases: embedding and searching. The searching part is as simple as possible, based on the observation from real life: in order to achieve the destination point one should choose the way, among others, which best follows the desired direction.

Certainly, one can experience blind corners or other configurations when going in the desired direction is not the optimal choice. This is what the embedding part of the algorithm is responsible for: rewriting the graph so that the simple approach would be successful. The graph embedding is based on an eigenproblem solution, which is computationally expensive, but calculated once per network life and can be done by an external unit.

The proposed walker algorithm can be seen as a modification of existing searching algorithms, like A-star; however, it comes with a new geometrical context. The main novelty proposed in the paper is to combine spectral graph embedding with simple route search, which makes the very simple search technique efficient and robust.

III. NAIVE WALKER ALGORITHM

At first, the searching part of the method is presented. The proposed routing algorithm is based on the following heuristics. Consider a graph shown in Fig. 1. Suppose one wants to reach node n5 starting from n1. Also assume that in n5 a big high tower is built, so that the direction one should

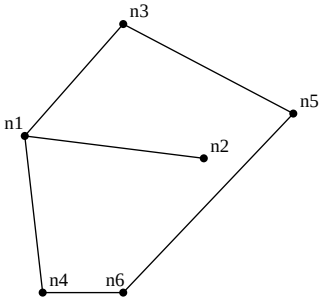

Fig. 1. Example graph for naive path search

follow is known. Thus, among all possible intermediate nodes: n3, n4 and n2 the last one seems the best choice.

Reaching n2, however, is a blind route and a person must move back and choose either n3 or n4. Thus, there is no guarantee that the choice of path in the direction closest to the destination is the optimal choice, but it is more or less how a human would behave.

The mathematical formulation of such behavior can be defined as follows: consider a vector n_i, pointing from the origin to the ith node of the graph. Then, standing at n_1, and knowing where the destination node n_D is placed, as the next chop, among nodes incident to n_1, one should choose n_i for which the dot product (denoted as $\circ$), l_i:

$$l_i = \frac{(n_D - n_1) \circ (n_i - n_1)}{||n_D - n_1||\,||n_i - n_1||} \tag{1}$$

is maximized. Provided no blind node is reached, the procedure is repeated until the target is reached.

This very simple algorithm has a number of drawbacks. It is possible not to reach a termination node (as discussed above), the route does not have to be optimal. However, the second part of the described technique is the preparation of graph in a manner which minimizes the risk of such incidents.

Without modification, it is possible that the naive walker will not reach the destination even if the path exists. There are several reasons for which the search procedure can terminate. The proposed remedy is also based on simple human heuristics: let us move back and try another track. This trivial observation makes the algorithm more complicated in implementation, but more reliable. The unmodified algorithm is stateless. Adding a possibility of correcting the path by moving back and changing another option requires keeping all the track in memory. The modified behavior can be described in the following points:

- If no further path towards the destination point exists, move to the previous node.
- Modify the graph so that the edge traversed twice (for and back) is excluded and not used in further search.
- Call the function with the previous point as a starting point.

This is a possible situation when the walker algorithm reaches dead end. Then, one must move back and remember this direction as forbidden in further search. The path search finishes in two cases: when the destination point is reached, or no further move is possible. In fact, this procedure is a guided deep search when the direction of the search is driven by the previously discussed parameter. The direct consequence is that the path will always be found, provided it exists.

III. 1. Embedding the network

Before the second part of the algorithm is presented, we should discuss an important property of Dijkstra's algorithm. It does not utilize physical placement of the nodes. It traverses the graph, but no geometrical information is used. The a-star algorithm utilizes the geographical information, but finally for regular structures it should give the same result as Dijkstra. Since in this paper we tend to replace the physical node placement with a virtual graph representing the network, real distances are no longer significant for our algorithm. The heuristics we present is based on direction rather than distance.

The second remark is that given a graph, there are infinite possibilities to draw (embed) it onto a plane. We propose the embedding for which a simple heuristics defined as "go towards the destination" works.

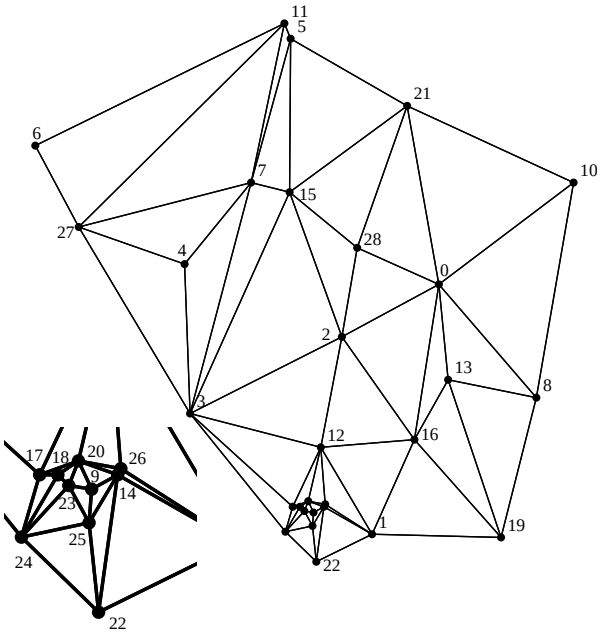

Fig. 2. Delaunay triangulation of 30 biggest cities in Poland

Consider a fully connected graph. In this case, all reasonable algorithms would point the direct path between nodes. Malfunction of a single edge would affect only one link, thus such a network is stable, consistent and well-balanced. For economic reasons, however, a telecommunication network is not a fully connected graph. Another issue related to a fully connected graph is the case when two paths are required for protected channels. In that case a direct link no longer exists and all previously mentioned issues must be considered.

Now, suppose the network is built according to Delaunay triangulation of the physical nodes. This creates a relatively smooth graph (compare Fig. 2, Delaunay triangulation of the 30 biggest cities in Poland). The optimal (shortest) route from the source to the destination is as close as possible to the straight line between these points. By constructing a network in such a manner, intuitively one can achieve well-balanced traffic, since optimal routes between points are likely to be disjoint and no "highways" appear.

Here arises the main concept of the two-phase algorithm: let us rewrite the existing graph so that it is smooth and regular. Let us work on a virtual graph where spatial coordinates of the nodes do not need to correspond to their physical placement. Thus redraw the graph and use the route search algorithm which results in a well-balanced network. Contrary to, say, logistics, for telecommunication traffic such a virtual graph can be successfully used, since for data transfer physical distance is not a factor determining the cost of the route. This is also the rationale for weighting all optical links as one, regardless its physical length.

IV. SPECTRAL GRAPH EMBEDDING

Graph embedding or, in other words, plotting a graph on a plane (if possible) is a demanding task and has countless applications, including efficient and ergonomic visualization [5].

An intuitive approach is based on treating edges as springs and by finding the stable configuration in terms of mechanical system. Another approach is spectral embedding, where the coordinates of the nodes are based on values in eigenvectors of the Laplacian matrix of a graph [7]. Consider the graph shown in Fig. 1. Its Laplace matrix is (2). Denoting ith component of jth eigenvector $v_{i,j}$, the coordinates of nodes for the embedded graph (Fig. 3) are the following: $n_1 = (v_{1,1}, v_{2,1})$, $n_2 = (v_{1,2}, v_{2,2})$, etc.

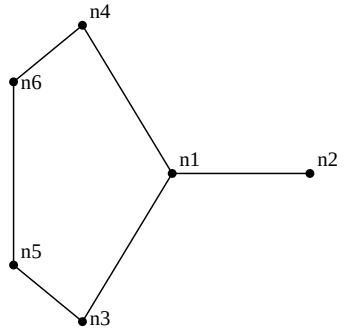

Fig. 3. Example graph for a naive path search, spectrally embedded graph from Fig. 1

$$\mathbf{L} = \begin{bmatrix} -3 & 1 & 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -2 & 0 & 1 & 0 \\ 1 & 0 & 0 & -2 & 0 & 1 \\ 0 & 0 & 1 & 0 & -2 & 1 \\ 0 & 0 & 0 & 1 & 1 & -2 \end{bmatrix} \tag{2}$$

Spectral embedding, despite some issues described later in this section, possesses several important properties:

- Strongly connected nodes (i.e. the pairs for which more different paths exist) are drawn closer to each other.
- Nodes connected with edge of higher weight are closer.
- Regular graphs looks regular.

The idea behind using an eigenvectors as nodes' coordinates comes from Laplace matrix, representing 2nd order differential operator [6]. Eigenvectors of this matrix have form $sin(n\pi x)$, thus, the ones related to smaller eigenvalues are slowly changing, and using them as a coordinates of nodes results placing the nodes along Lissajous curve. Although the graph Laplace matrix (2) has a different form, it shares some properties of matrix representing Laplace operator: it is diagonally dominant, sparse and semidefinite.

IV. 1. Dealing with lined up nodes

For regular graphs it is likely that subsequent nodes are placed in the same point on the plane, which is not a desired behavior. An example of this issue can be seen in Fig. 6 and 4, which show embedding of $K_{3,3}$ and K_5 graphs, respectively. As seen, nodes x and y are in the same place. Similarly, it is the case for k1 and k2 nodes for K_5 graph. Imagine for a moment that K_5 graph is embedded into 3D space, not in a plane. Then k1 is above the plane and k2 is below, which results in two joined tetrahedrons, a perfectly regular structure. However, the scope of this paper is planar embedding, and in such a case, node collocation on a plane should be avoided. To overcome this issue, eigenvectors with related higher eigenvalues are used. In the presented tests, coordinate of ith node (x_i, y_i) is found as follows:

$$\begin{aligned} x_i &- v_{i,2} + cv_{i,4}, \\ y_i &- v_{i,3} + cv_{i,5}. \end{aligned} \tag{3}$$

In other words, the coordinates are a combination of 2nd and 4th for x component. and 3rd and 5th vector for y component. Coefficient c in (3) is a correction factor, which should be relatively small in order not to disturb original embedding significantly. In Fig. 7 and 5, the corrected versions of $K_{3,3}$ and K_5 are shown.

IV. 2. Concluding remarks

As presented, the route searching algorithm is split into two phases: embedding and searching. The embedding is done once when the network is put into service, or when significant changes are applied. It occurs rarely and the spectral embedding, which requires a solution of the matrix eigenproblem, can be done using resources other than units built into network nodes.

The searching part is computationally cheap, it is n dot products in 2D Cartesian space per step, where n is the order of the current node. When, like in real applications, two paths, edge disjoint or node disjoint should be found,

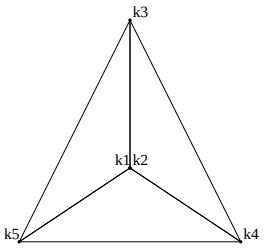

Fig. 4. Embedded K_5 graph, withoutout correction

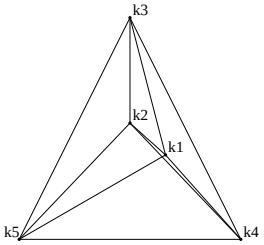

Fig. 5. Embedded K_5 graph, with correction

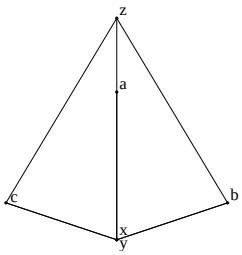

Fig. 6. Embedded $K_{3,3}$ graph, without correction

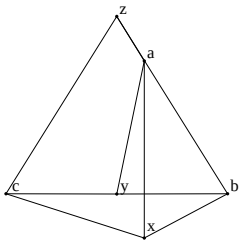

Fig. 7. Embedded $K_{3,3}$ graph, with correction

a number of "walkers" can be run in different directions (first best direction, second best direction, etc.). When a number of "walkers" was equal to the node's order, eventually the algorithm became equivalent to deep-search. In this context it could be compared to A-star [13], which also recursively traverses the graph, driven by estimation of remaining distance. The presented method is, on the contrary, driven by direction. Exact steps for naive implementation can be written as follows:

1. Let the current node n_0 be the starting one.
2. Calculate inner products between normalized vectors $n_i - n_0$ and $n_D - n_0$, where n_D is a destination node.
3. Choose n_i for which the results are the largest, remove n_0 from graph and skip to step 1.

As the naive walker can be spawned multiple times, in our tests a small number, like 5 to 7, of "walkers", were run. Each of them traversed the graph. If it was about to reach a node visited by another instance of walker, depending on the configuration, it changed direction of the just terminated one. Not all walkers must succeed, i.e. reach the destination node. However, a number of routes should be found. Among several possible routes one can select an edge disjoint pair or a node disjoint pair. It was also possible to equalize the length of two paths, which is advantageous in protected transmission, when a signal is simultaneously transmitted by two separate tunnels. The steps for path calculation when multiple walkers are used are the following:

1. Set the maximum number of walker instances M.
2. Let the current node n_0 be the starting one.
3. Find all adjacent n_i nodes, $i > 0$ to n_0.
4. If M is not exceeded, run a naive walker for every adjacent node n_i. Update the walker instance counter.

For large M the above scheme would become equivalent to a deep search. However, for a reasonably small number of instances it rather sends a scout. Among all successful scouts, pairs with no common edges are built and, depending on the objective, the first combination or the shortest (in terms of edges count) is used. The latter makes sense for bigger M.

The computational complexity of the algorithm is in fact irrelevant, provided the graph embedding is done once, when the network is running into service. The embedded graph operations involve expensive resources; however, the latter can be calculated externally and distributed among network slightly cheaper nodes. The walker itself is computationally cheap.

V. EXAMPLES OF CHANNEL PROVISIONING

As discussed earlier, in telecommunication applications one can neglect the physical length of a path. The parameter of interest is the virtual length of a tunnel, measured in the number of chops signal must traverse. The physical length of the fiber has minor importance and can be neglected in most situations. When the signal crosses network element, which is represented as a node in a graph, it can be converted into an electrical signal or routed using pure optics. Both operations involve expensive resources, however the latter is slightly cheaper.

On the other hand, the equipment kept in safe areas is much less likely to be accidentally damaged compared to optical fibers. For this reason a provider usually finds two disjoint routes between nodes in order to provide a restoration path. There are two types of restoration: the spare channel is activated on demand, which results in several seconds

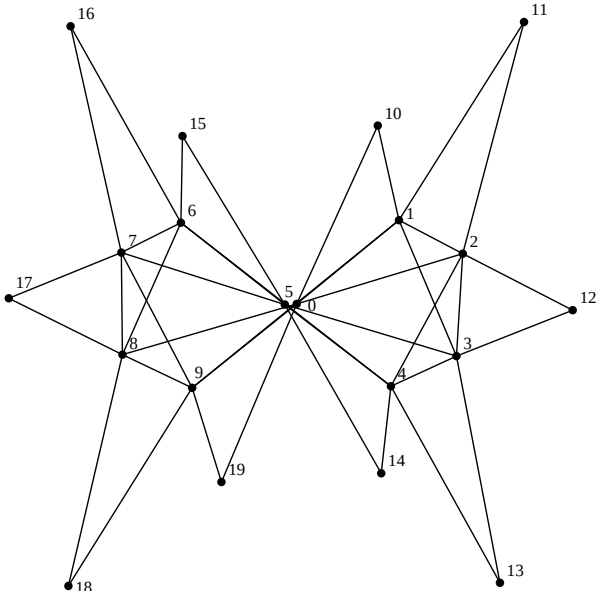

Fig. 8. Star like graph layout

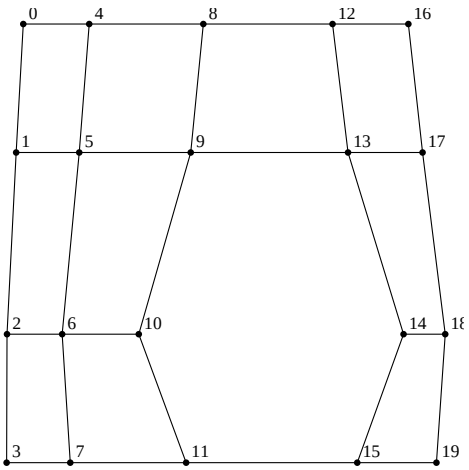

Fig. 9. Grid based graph

of data loss, and the second when two tunnels are active and the signal goes simultaneously. The receiver decodes however one of them. In case of failure it switches to the second one. Therefore, the restoration path should also be optimized in terms of length and resource usage. Moreover, it is a good idea to make these two paths equal in length due to delays and buffering in egress and ingress nodes.

V. 1. Testing methodology

We present the results obtained with a naive walker combined with spectral embedding compared to Dijksta shortest path, as the latter is used in routing in optical networks [12]. The parameter of interest is a distribution of traffic over the network, measured in a number of paths held in a particular graph edge (optical fiber).

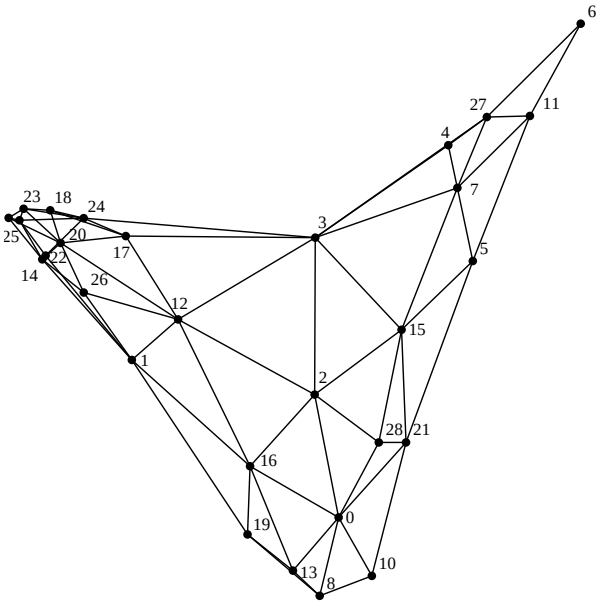

Fig. 10. Graph build using connections among Polish cities

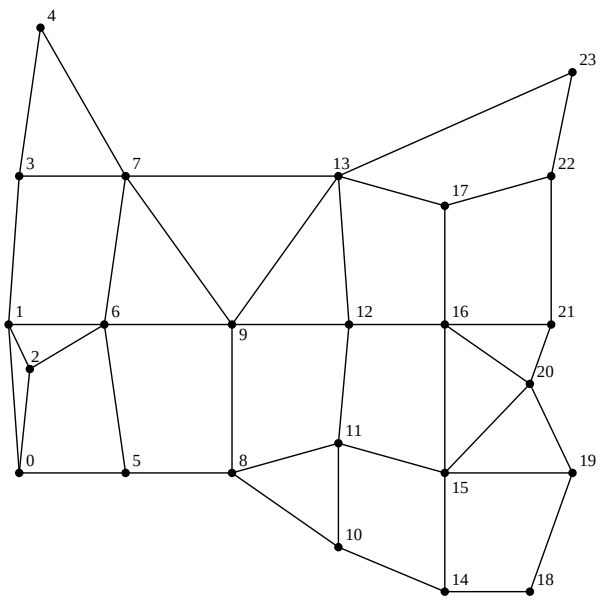

Fig. 11. Original layout of 24 node USA network

Network topologies are: a simple square grid with excluded several nodes shown in Fig. 9, a star-like structure (Fig. 8), a graph of selected Polish towns and, finally, a 24-node network which operates in the USA [1]. The graph based on Polish towns (Fig. 10) is a rewritten version of the network shown in Fig. 2. The last network is shown in Fig. 12, and geographical placement of nodes is shown in Fig. 11.

Two first graphs are regular, thus the traffic should be well-balanced regardless of the used algorithm. Polish towns were built as a Delaunay triangulation of nodes, thus

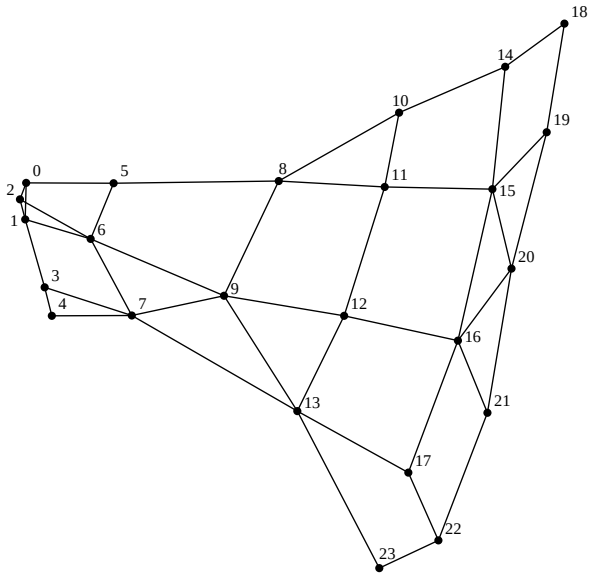

Fig. 12. 24 node USA network

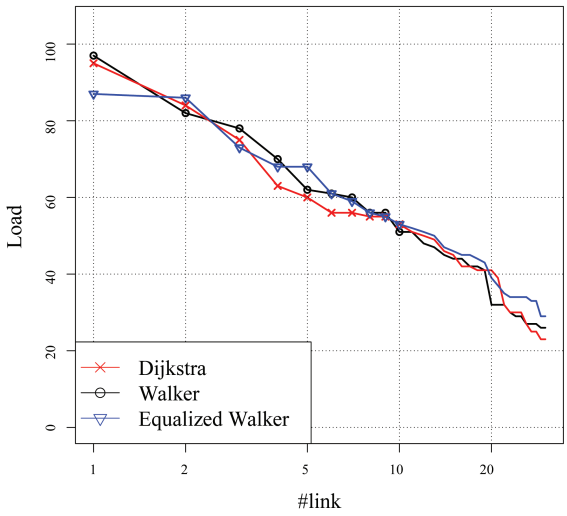

Fig. 14. Load profile of the grid based network shown in Fig. 9

this graph is also very regular. Moreover, the number of equivalent (i.e. having the same length) paths is reduced, compared to a square-based grid. Again, the 24-node network is square-based, much more irregular compared to the square grid.

In each test a few walkers were spawned. The walkers were not aware of any network state, like the existent load, etc. Therefore, the results are fully deterministic. Some global optimization could be possible, but in this case parameters of a path depend on the order of provisioning.

Each of the walkers yielded a single path, and the next step was choosing two paths from the set of solutions. This pair was chosen so that the overall length was minimized or, in the alternative approach, the difference in the number of chops for both tunnels was minimum or, if possible, zero. To this end, the latter approach is named as equalized walker.

The constraint was the following: no pair of the paths should have a common edge. Note that the choice between several paths can lead to ambiguity. However, when all edge weights are set to unity, Dijkstra algorithm is also not unique. In the tests we have not analyzed this issue in detail.

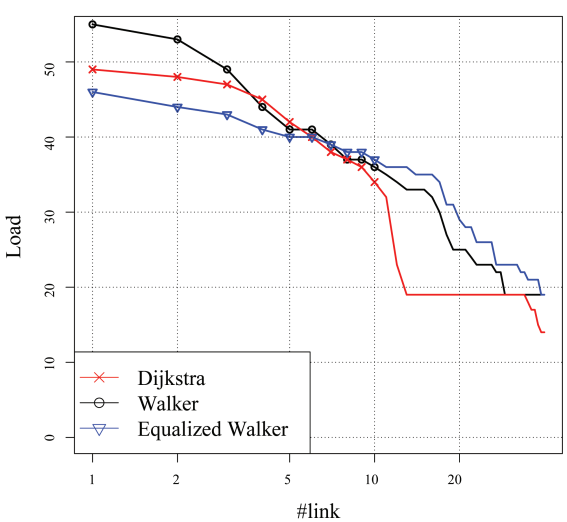

Fig. 13. Load profile of the star-like network shown in Fig. 8

V. 2. Grid based network

As noted in the introduction to this section, a grid-based network is subject to ambiguity in path calculation due to existence of multiple possible paths of the same length (cost). Depending on how the walker paths were selected, one can decrease or increase the maximum link load. In terms of load balancing one can both improve and worsen resource usage, thus this network is very sensitive to how the resources are provisioned. This behavior is shown in Fig. 14: the result obtained with Dijkstra is between two walker formulations. Detailed results are shown in Tab. 1.

V. 3. Star structure

For a star-like structure shown in Fig. 8, results of simulation, namely searching all-to-all tunnels are presented in Tab. 2 and visualized in Fig. 13. In this case a simple walker does not improve the performance over Dijksta, even the load of the most used link is increased by approximately 10 percent (from 49 to 55). With additional constraint added to walker routine, the results were significantly improved. At this run, the objective was to equalize primary and restoration path lengths.

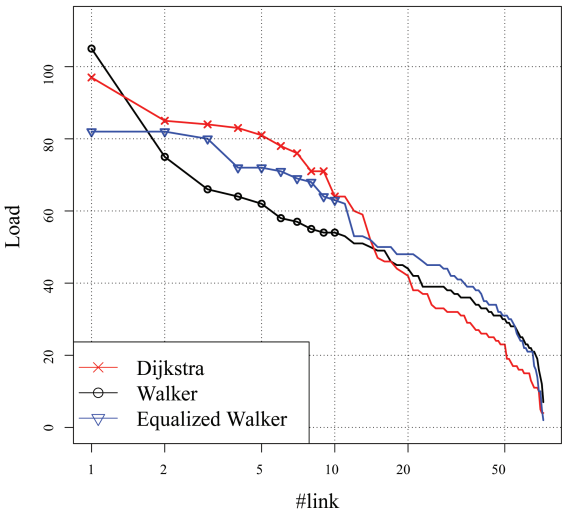

Fig. 15. Load profile of the "Poland" network shown in Fig. 10

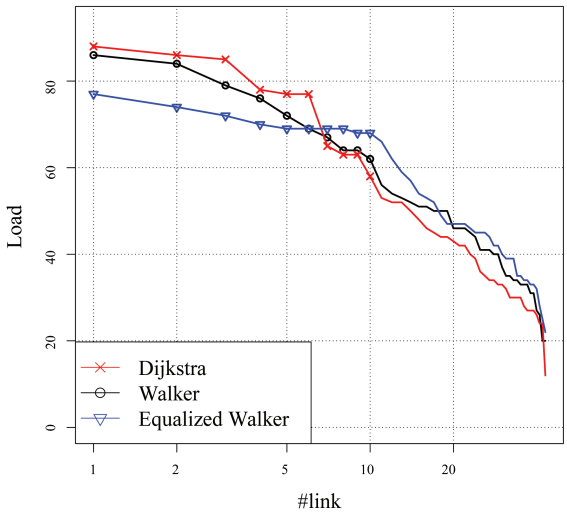

Fig. 16. Load profile of the 24-node USA network shown in Fig. 12

V. 4. Network based on Polish cities

The third network chosen as an example is very regular, thus one can expect good performance of all algorithms. Depending on how the tunnel pair was selected, one can slightly increase or decrease the maximum load. However, in both cases the traffic was more uniformly distributed over network: it was moved to less used links, as seen in Fig. 15. What is worth pointing out, for this network it was possible to almost exactly equalize primary and restoration path length, as shown in the last row of Tab. 3.

V. 5. 24 node USA network

The last presented example is a real network [1]. This network is both regular and square based which, as mentioned above, leads to ambiguities in path calculation. However, rewriting the graph in a way described in this paper resulted in a shape where the directions between egress and ingress are unique and straightforward, which is visible in the performance of the walker algorithm that bases on directions (Fig. 16).

Tab. 1. Comparison of network parameters for Dijkstra and walker path search algorithm for network from Fig. 9

	Dijkstra	Walker	Equalized walker
Max. link load	95	87	96
Avg. link load	46.13	48.6	48.6
Avg. primary path len.	3.06	3.74	3.63
Avg. restoration path len.	4.22	3.93	4.05
Avg. difference in primary and restoration path len.	1.15	0.52	0.52

Tab. 2. Comparison of network parameters for Dijkstra and walker path search algorithm for network from Fig. 8

	Dijkstra	Walker	Equalized walker
Max. link load	49	55	46
Avg. link load	24.6	28.8	30.2
Avg. primary path len.	2.28	2.7	3.12
Avg. restoration path len.	2.88	3.36	3.23
Avg. difference in primary and restoration path len.	0.6	1.06	0.1

VI. SUMMARY

The deterministic algorithm for path calculation has been presented. It combines very simple "naive walker" heuristics with spectral graph embedding. The graph embedded according to eigenvectors of its Laplace matrix is usually regular and has a reasonable shape, which makes the very simple walker model useful and successful. The two-phase nature of the algorithm allows one to split the computations into two separate phases. A solution of the eigenproblem, which is computationally expensive, is calculated once per network and the result can be propagated into network nodes. A direction-based search,which requires only sorting and dot product calculation is inexpensive and can be easily calculated on a small units.

The algorithm can be a simple alternative to those currently used. The proposed technique, compared to existing algorithms, can be computationally expensive. However,

Tab. 3. Comparison of network parameters for Dijkstra and walker path search algorithm for network from Fig. 10

	Dijkstra	Walker	Equalized walker
Max. link load	97	82	102
Avg. link load	34.3	39.7	38.9
Avg. primary path len.	2.64	3.74	3.453
Avg. restoration path len.	3.44	3.84	4.458
Avg. difference in primary and restoration path len.	0.807	1.105	0.005

Tab. 4. Comparison of network parameters for Dijkstra and walker path search algorithm for network from Fig. 12

	Dijkstra	Walker	Equalized walker
Max. link load	88	77	85
Avg. link load	44.9	49.3	52.3
Avg. primary path len.	2.99	2.23	4.01
Avg. restoration path len.	4.0	4.45	4.14
Avg. difference in primary and restoration path len.	1.01	1.42	0.17

the path calculations and graph shape optimization can be performed in advance, once for the whole network. Moreover, no network resources need to be used for this task. In that sense the solution is similar to how neural networks are implemented in popular devices. Networks' training is expensive, but using the trained one is not. Although it is not formally guaranteed, path calculation using a naive walker results in better network balance with no global optimization, which means that the path calculation is independent of the network state.

The usefulness of this method was presented for several real-life and similar to real-life networks. The assumptions and restrictions were based on real telecommunication scenarios.

References

[1] J. de Santi, A. Drummond, N. de Fonesca, X. Chen, A. Jukan, *Leveraging Multipath Routing and Traffic Grooming for and Efficient Load Balancing in Optical Networks*, IEEE Trans on Optical Networks and Systems, 2989–2993 (2012).

[2] A. Rahman, N.M. Sheikh, *Modified Bidirectional Reservation on Burst drop with Dynamic Load Balance in Optical Burst Switching*, 11th Int Conference on Telecommunications – ConTel 2011, Graz Austria.

[3] J. Zhang, S. Wang, K. Zhu, L. Sone, D. Datta, Y. Kim, B. Mukherjee, *Optimized Routing for Fault Management in Optical Burst-Switched WDM Networks*, IEEE Journal of Selected Areas in Communication **25**(6), 111–120 (2007).

[4] M. Chamania, A. Jukan, *A Survey of Inter-Domain Peering and Provisioning Solutions for the Next Generation Optical Networks*, IEEE Communication Surveys and Tutorials, **11**(1), 33–51 (2009).

[5] P. Idziaszek et al, *Visualisation of Relational Database Structure by Graph Database*, CMST **22**(4), 217–224 (2016).

[6] C.D. Meyer, *Matrix Analysis and Applied Linear Algebra*, SIAM 2000.

[7] B. Luo, R. Wilson, E.R. Hancock, *Spectral embedding of graphs*, Pattern Recognition, **36**(10), 2213–2230 (2003).

[8] S. Hardy, *Romtelecom taps ADVA FSP 3000 for multiple networks*, Lightwave, June 2011, https://www.lightwaveonline.com/articles/2011/06/romtelecom-taps-adva-fsp-3000-for-multiple-networks-123097013.html, (access 20.11.2018)

[9] T. Eilam-Tzoreff, *The disjoint shortest paths problem*, Discrete Applied Mathematics, **8**(2), 113–138 (1998).

[10] L, Guo, Y. Deng, K. Liao, Q. He, T. Sellis, Z. Hu, *A Fast Algorithm for Optimally Finding Partially Disjoint Shortest Paths*, Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18).

[11] *Ultra low latency networks.* Application Note, ADVA Optical networking 2012 (available online, www.advaoptical.com).

[12] OSPF Version 2, protocol specification, RFC 2328.

[13] R. Dechter, J. Pearl, *Generalized best-first search strategies and the optimality of A**, Journal of the ACM **32**(3), 505–536 (1985).

Michał Wiktor received his PhD in Computational Electromagnetics in 2006 at Gdansk University of Technology. He worked as a researcher and lecturer in the Department of Statistics in Gdansk Medical University. His research included medical imaging and influence of electromagnetic fields with living organisms. In 2011 he joined ADVA Optical Networking, where he worked as a software engineer. Currently he works in the Artificial Intelligence Department at Intel Poland. His interests include mathematical modelling and functional programming.