Electronic Journal of Graph Theory and Applications **5** (2) (2017), 221–240

Electronic Journal of
Graph Theory and Applications

"Transit data"-based MST computation

Thodoris Karatasos[a], Evi Papaioannou[b]

[a] *University of Patras, GR26504, Rion, Greece*
[b] *University of Patras & CTI "Diophantus", Patras University Campus, Building B, GR26504, Rion, Greece*

tkaratasos@upatras.gr, papaioan@ceid.upatras.gr

Abstract

In this work, we present an innovative image recognition technique which is based on the exploitation of transit-data in images or simple photographs of sites of interest. Our objective is to automatically transform real-world images to graphs and, then, compute Minimum Spanning Trees (MST) in them.

We apply this framework and present an application which automatically computes efficient construction plans (for escalator or low-emission hot spots) for connecting all points of interest in cultural sites, i.e., archaeological sites, museums, galleries, etc, aiming to to facilitate global physical access to cultural heritage and artistic work and make it accessible to all groups of population.

Keywords: minimum spanning tree, image recognition, transit data, global physical access, culture
Mathematics Subject Classification : 05C85, 05C90
DOI:10.5614/ejgta.2017.5.2.6

1. Introduction

The Universal Declaration of Human Rights provides that "everyone has the right to freely participate in the cultural life of the community, to enjoy the arts and to share in scientific advancement and its benefits" [27]. Indeed, one of the key requirements for an inclusive and sustainable society is that everyone should be able to participate in and enjoy the social, economic and cultural assets of that society. Historic places are a significant asset, a unique and irreplaceable resource which reflects a rich and diverse expression of past societies and forms an integral part of local, regional and national cultural identity.

Received: 18 April 2017, Revised: 30 August 2017, Accepted: 8 September 2017.

However, over the years, persons with disabilities, either temporary or long-term, have faced barriers to realizing their right to participation in the cultural life of the community due to inaccessibility of cultural programmes, premises and venues. For some people, barriers exist which make visiting historic places difficult or sometimes impossible. For example, during visits in cultural sites, i.e., archaeological sites, museums, galleries, etc., not all visitors enjoy equal opportunities for easy physical access to all site places and offered facilities. This could be due to particular constraints like lack of escalators or elevators, lack of wireless access to networked facilities during the visit or even difficulty to reach such existing facilities.

Making heritage sites more accessible in an appropriate and sensitive manner can increase awareness and appreciation of its cultural, social and economic value. It assists in meeting society's requirement to protect its architectural heritage, while also meeting the need to provide equal access for all, as far as is practicable [26]. In order to facilitate physical access to cultural heritage and artistic work and make it accessible to all groups of population, a reasonable approach to correct such inefficiencies could be the automatic design of physical paths joining all places of a cultural site. Lowering the induced construction cost would make such a proposal cost-efficient and thus realistic and worth-implementing. Graph theory, i.e., the study of structural properties of graphs, has emerged to a branch of mathematics providing deep understanding and helpful insight not only for scientific research questions but also for a wide range of real-world problems. In this context, our motivating question has been whether we can exploit existing results from graph theory and relevant algorithmic methods in order to facilitate efficient global access to cultural sites.

Towards this direction, an important objective consists in suggesting the minimum length of total - escalator or low-emission hot spot - units required to cover the site. Assuming that each cultural site is internally connected, the problem of efficient global access to cultural sites can be abstracted as a well-known problem from graph theory, namely the Minimum Spanning Tree (MST) problem. Coupling the MST problem with an innovative image recognition technique based on the exploitation of transit-data in input images, we designed and implemented, using the MATLAB programming environment, an application which automatically computes efficient construction plans (for escalator or low-emission hot spots) for connecting all points of interest in cultural sites. Providing as input an image or a simple photograph of the site in question, the administration of cultural sites can quickly obtain an efficient construction plan which can be used to provide an escalator construction plan of minimum total length and can also be further used for placing low-emission routers for wireless guiding services and transmission of cultural information to visitors. To the best of our knowledge no similar application has been suggested in the recent relevant literature.

The rest of the paper is structured as follows: in Section 2 we present the MST problem as well as relevant algorithmic solutions. In Section 3, we provide design and implementation details for our application. We conclude and present future plans in Section 4.

2. The MST Problem

The MST problem is one of the simplest and best-studied optimization problems. It consists in finding a spanning tree of an undirected, edge-weighted, connected graph, such that the sum of the

weights of the selected edges is minimum.

The significance of the MST problem stems from the plethora of its applications as well as from the existence of computationally efficient methods which makes it practical to solve MST for large graphs. The MST problem has direct applications in a wide range of problems involving network design. Furthermore, many indirect applications of the MST problem stem from the fact that it is essentially underlying in several other problems. More precisely, the MST problem has been directly used to model and solve design problems in computer and communication networks and also in transportation and power supply networks. It has also emerged as a crucial component in the context of other seemingly less related problems like clustering and classification. Furthermore, the MST problem usually forms a subproblem in the solution of problems like for instance the well-known Travelling Salesman Problem, the matching problem, etc.

In the following sections, we formally state the MST problem, overview algorithmic solutions from the relevant literature describing in detail three simple and efficient classical algorithms for computing minimum spanning trees and highlight some of the diverse multidisciplinary applications of the MST problem.

2.1. Problem statement

The MST problem is one of the most typical and well-known problems of combinatorial optimization. Algorithms for its solution, though simple, have generated important ideas of combinatorics and have played a key role in the design of algorithms [7, 9, 24, 25]. A simple, intuitive statement of the problem provided by Horowitza and Sahni [10] is as follows.

"*Given such a weighted graph one would then wish to select for construction a set of communication links that would connect all cities and have minimum total cost or be of minimum total length. In either case the links selected will have to form a tree (assuming all weights are positive). In case this is not so, then the selection of links contains a cycle. Removal of any of the links on this cycle will result in a link selection of lower cost connecting all cities. We are therefore interested in finding a spanning tree of G with minimum cost. (The cost of the spanning tree is the sum of the costs of the edges in that tree.)*"

2.2. General solution and classical algorithms

Traces of the MST problem date back to 1926 when Borůvka first studied primarily an Euclidean version of the problem related to planning of electrical transmission lines [3], and suggested an efficient algorithm for the general version of the problem [4]. Borůvka's work was further extended by Jarník [11], again in a mostly geometric setting, who suggested another efficient algorithm. However, when computer science and graph theory started forming in the 1950s and the Spanning Tree Problem was one of the central topics of the flourishing new disciplines, previous work was not well-known and the algorithms were rediscovered several times. In the next 50 years, several significantly faster algorithms were suggested, ranging from the algorithm by Fredman and Tarjan [8], over algorithms with inverse-Ackermann-type complexity by Chazelle [5] and Pettie [21], to an algorithm by Pettie and Ramachandran [22] whose time complexity is provably optimal. Frequently, the most important ingredients were advances in data structures used to represent the graph [14].

A naive method for solving the MST problem in a given graph G could consist in exhaustively computing all spanning trees of G and then selecting that (or those) of minimum weight. However, even if an appropriate way of listing all computed spanning trees were available, this would make a quite inefficient approach due to the potentially large number of spanning trees required to compute.

Investigating the relevant literature, we can easily see that well-known, classical, simple algorithms for the MST problem use a tree growing approach. This approach actually detects a key property of the MST determining whether an edge should be included in it, and subsequently exploits this property to build up the MST incrementally, in an edge by edge fashion, including appropriate light edges and excluding appropriate heavy ones. The approach is greedy in the sense that at each step, the best possible edge is selected for inclusion in the MST without producing a cycle in the subgraph constructed so far or for exclusion from the MST without disconnecting the graph.

While there is a long list of algorithms for the MST problem, we decided to include a detailed description for three classical algorithms for constructing Minimum Spanning Trees, namely Borůvka's algorithm [4], Kruskal's algorithm [13] and Prim's algorithm [23]. Borůvka's algorithm has been the basis for several improved algorithms for the MST problem. Regarding the algorithms of Kruskal and Prim, they both proceed by successively adding edges of smallest weight from those edges with a specified property that have not already been used. The main difference is the criterion used to select the next edge or edges to be added in each step. They are particularly simple and in fact solve the same problem by applying the greedy approach in two different ways and both always yield an optimal solution.

2.2.1. Borůvka's algorithm

In 1926, Otakar Borůvka described methods for constructing minimum spanning trees in work relating to the construction of electric power networks [3]. In short, the algorithm begins by first examining each vertex and adding the cheapest edge from that vertex to another in the graph, ignoring already inserted edges, and continues joining these groupings in a similar manner until a tree spanning all graph vertices is generated. More precisely, given a connected, undirected graph G, Borůvka's algorithm proceeds as follows [18].

1. Initially all edges of G are uncolored and let each vertex of G be a trivial blue tree.
2. Repeat the following step until there is only one blue tree.
3. For every blue tree T, select the minimum-weight uncolored edge incident to T. Color all selected edges blue.

This algorithm is called parallel merging or forest growing. It requires $O(\log |V|)$ iterations while each iteration needs $|E|$ steps. Thus, the algorithm runs in time $O(|E| \log |V|)$, where $|E|$ is the number of edges and $|V|$ is the number of vertices in G. The speed up of this algorithm has been intensively studied in the recent literature. For example, Klein and Tarjan [12] combined Borůvka's algorithm with filtering based on random sampling to obtain a randomized algorithm which runs in linear expected time. The best known (deterministic) MST algorithm [5] is also partially based on Borůvka's algorithm achieving a running time of $O(E\alpha(E, V))$, where α is the inverse of the Ackermann function. The main underlying idea in all these randomized and

deterministic algorithms lies in combining steps of Borůvka's algorithm, in order to decrease the number of components remaining to be connected, with steps of a different type that reduce the number of edges between pairs of components.

2.2.2. Kruskal's algorithm

The algorithm has been attributed to Joseph Kruskal who suggested it in 1956 directly relating it with the "shortest spanning tree" problem [13]. However, basic ideas of the algorithm were actually described much earlier in works that had not received the necessary attention due to indirect references to the MST problem.

Kruskal's algorithm starts with an edge in the graph with minimum weight and builds the spanning tree by successively adding edges one by one into a growing spanning tree. It processes the edges in order of their weight values, from smallest to largest, including into the growing MST each edge which does not form a cycle with edges previously added. It stops after $|V| - 1$ edges have been added. Kruskal's algorithm computes the MST of any connected edge-weighted graph with E edges and V vertices in time proportional to $E \log E$ (in the worst case) since sorting is the most time consuming operation. More precisely, given a connected, undirected graph G, Kruskal's algorithm proceeds as follows [13].

1. Sort the graph edges by their weights.
2. Add edges to the MST starting from the edge with the smallest weight and proceeding to the edge of the largest weight.
3. Only add edges which do not form a cycle (i.e., edges which connect only disconnected components).

It is worth noting that Kruskal's the algorithm is best in situations where the edges are given in sorted order or can be sorted fast, like when the weights are small integers, making it possible to use efficient sorting algorithms, or where the graph is sparse.

2.2.3. Prim's algorithm

The algorithm was originally presented by the Czech mathematician Vojtěch Jarník in 1930 [11], though, it became well-known when it was rediscovered in 1957 by Robert Prim [23].

Prim's algorithm constructs a minimum spanning tree incrementally, in a step-by-step fashion via a sequence of expanding subtrees. The initial subtree of the sequence consists of a single vertex selected arbitrarily from the set V of the vertices of the given graph. In each successive step, the algorithm expands the current tree greedily by simply adding to it the nearest vertex not in the tree. The distance of such a vertex is determined by the weight of the edge connecting it to the tree. In the case of at least two candidate nearest vertices, ties can be broken arbitrarily. The algorithm terminates when all vertices of the graph have been included in the spanning tree. Since the algorithm expands a tree by exactly one vertex during each step, the total number of required steps is $n - 1$, where n is the number of vertices of V. The tree generated by the algorithm is obtained as the set of edges used for the tree expansions.

More precisely, the algorithm maintains two disjoint sets of vertices: one containing vertices that are in the growing spanning tree and another containing vertices not in the growing spanning tree. Then, it selects the lowest-cost vertex which is connected to the growing spanning tree but is

not in the growing spanning tree and inserts it into the growing spanning tree. In order to avoid the creation of cycles, the algorithm marks the vertices which have been already selected and considers only those vertices that are not marked.

Since each vertex is considered only once, the time complexity of the Prim's Algorithm is $O((V + E) \log V)$. Fast parallel algorithms have been obtained for arbitrary sparse graphs by combining Prim's and Borůvka's algorithms [2]. These parallel algorithms achieve speedup when compared with the best sequential algorithm. In one of them, the key idea is that when run on a single processor the parallel algorithm behaves as Prim's, when run on n-processor network it becomes Borůvka's and it runs as a hybrid combination in a p-processor network, where $p \in (1, p)$. Another parallel approach finds a minimum spanning tree/forest using coordinated instances of Prim's sequential algorithm running on the graph's shared data structure.

2.3. Applications

MST is a fundamental problem with diverse applications. A wide variety of problems are solved by finding a spanning tree in a weighted graph such that the sum of the weights of the edges in the tree is a minimum.

The MST problem has obvious applications in the design of computer and communication networks, in the context of practical problems like VLSI layout, wireless communication and distributed networking. It can also be directly applied to the design of power networks, like water supply networks and electrical grids, leased-line telephone networks, wiring connections, links in a transportation network, piping in a flow network, etc [9, 15].

For example, consider a company with several offices in different cities which must be interconnected via leased telephone lines. A provider charges different costs to connect different pairs of cities. The objective of the company is to lease an appropriate set of lines so that all offices are connected at a minimum total cost. Then, this set of lines should be a spanning tree in the graph induced by the offices. Broadcasting in computer networks is another practical MST-based application important in data networking. An indicative relevant example is multicasting over IP (Internet Protocol) networks. To send data from a source computer to multiple receiving computers, each of which is a subnetwork, data could be sent separately to each computer. This type of networking, called unicasting, is inefficient, because many copies of the same data are transmitted over the network. To make the transmission of data to multiple receiving computers more efficient, IP multicasting is used. With IP multicasting, a computer sends a single copy of data over the network, and as data reaches intermediate routers, the data are forwarded to one or more other routers, computers that are dedicated to forwarding IP datagrams between subnetworks in a network, so that ultimately all receiving computers in their various subnetworks receive these data.

Furthermore, the MST problem offers a method for solution to other problems to which it applies less directly, such as network reliability, surface homogeneity test, picture processing, handwriting recognition of mathematical expressions, automatic speech recognition, clustering (e.g., clustering points in the plane, clustering gene expression data, graph-theoretic clustering) and classification problems (like comparing ecotoxicology data) [9].

Recent problems in biology and medicine such as medical imaging [1] and proteomics [19] as well as problems related to national security and bio-terrorism such as detecting the spread of

toxins through populations in the case of biological/chemical attacks [6] have also been modelled and studied on the basis of the MST Problem.

The MST problem often occurs as a subproblem in the context of other problems. For example, MST algorithms are used in several exact and approximation algorithms for the Travelling Salesman Problem (whose objective is to find the shortest path that visits each point at least once), the multiterminal flow problem, the matching problem, etc [9].

3. "Transit-data"-based MST generation for cultural sites

We implemented our application using the MATLAB programming environment. In particular, we used MATLAB version 8.5.0.197613 (R2015a) running on a Windows 7 machine with an AMD Athlon (tm) 64 X2 Dual Core Processor 4200+ CPU (2.2Ghz) and a RAM of 2GB. MATLAB is a numerical computing environment and fourth-generation programming language which allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran. Although it was intended primarily for numerical computing, it also allows symbolic computing, graphical multi-domain simulation and model-based design for dynamic and embedded systems. It has been widely used in academia and industry by users coming from various backgrounds of engineering, science and economics. MATLAB was first adopted by researchers and practitioners in control engineering, and quickly spread to many other domains. It is now also used in education, in particular the teaching of linear algebra and numerical analysis, and is very popular amongst scientists involved in image processing [16]. Our application receives as input an image file of a cultural site where points of interest have been appropriately marked and connected (Fig. 1). The image is then processed and automatically transformed into a working graph, where points of interest correspond to vertices and there is an edge between two vertices if the corresponding points of interest are actually connected by means of a physical link. In the resulting graph, vertices indicated with a yellow number have been identified and stored together with the edges connecting them in the corresponding graph (Fig. 2).

Since there is a physical path connecting every pair of points of interest in the cultural site, the resulting graph is connected. Subsequently, edges of the graph are assigned weights which are provided as input by the application user; edge weights depicted with red numbers could reflect physical distance or construction cost and produce a weighted graph (Fig. 3).

Using Prim's algorithm, a minimum spanning tree is computed in this weighted graph which is then visually presented on the output *.png* image file (Fig. 4).

The computed MST provides a plan of minimum total cost for escalator construction based on the given weights. Furthermore, lengths of the MST edges indicate the coverage range requirements for wireless hot spots placed on MST nodes. Since edges of low weight are in principal selected for the MST, coverage requirements are also determined accordingly leading to low emission installation patterns.

3.1. Automatic image-to-graph transformation

The flow diagram of the automatic image-to-graph transformation is shown in Fig. 5.

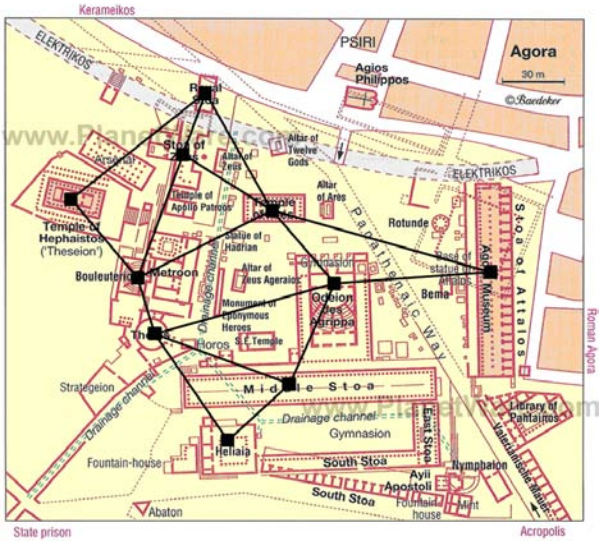

Figure 1. A sample input for the archaeological site of Agora in Ancient Athens, Greece

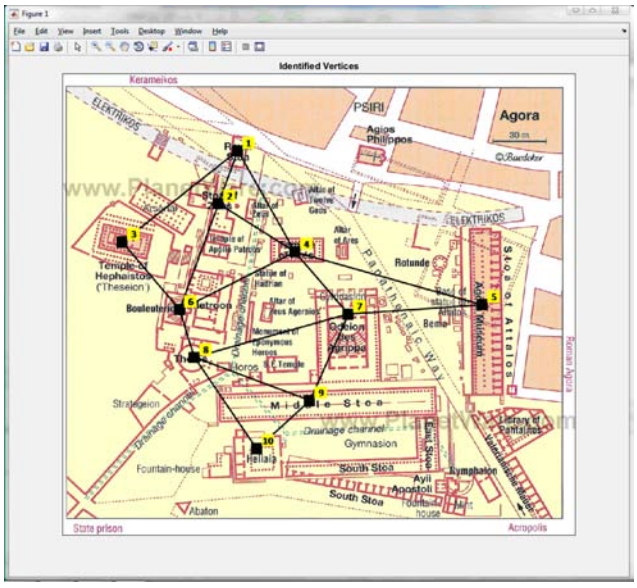

Figure 2. The resulting graph

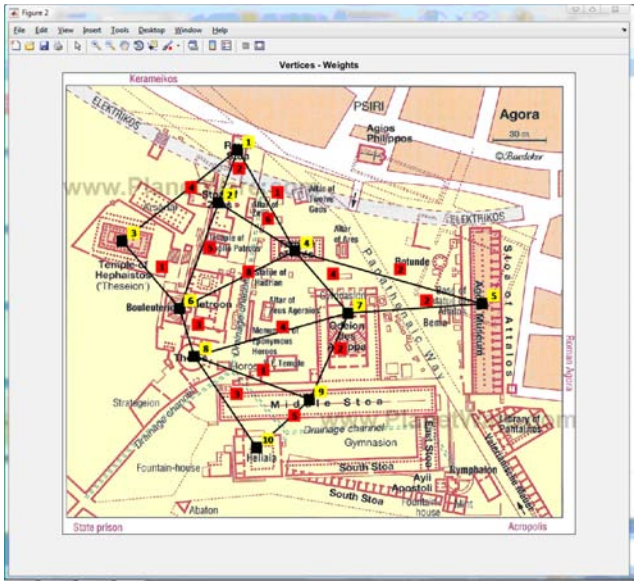

Figure 3. The resulting graph

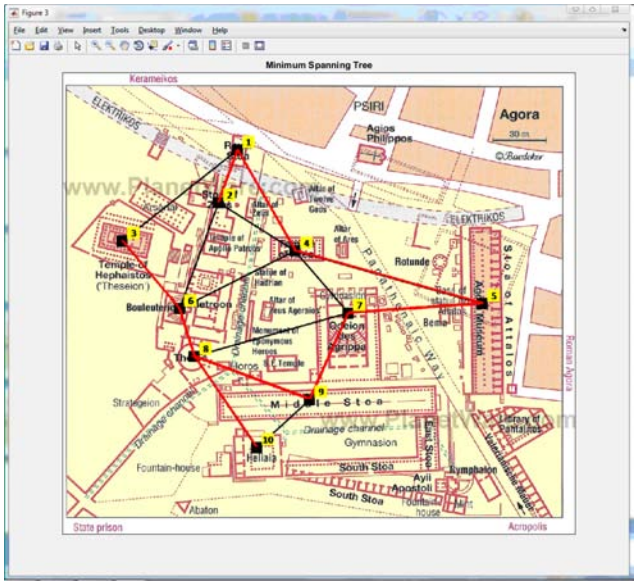

Figure 4. The resulting graph

Automatic image-to-graph transformation

- **Input:** Image (.png) with marked points of interest as vertices and connections as edges between them
- **Process**
  1. Image Loading
  2. Vertex Recognition
  3. Edge Recognition
  4. Edge-Weight Assignment
- **Output:** Adjacency Matrix for the resulting weighted graph

Figure 5. Automatic image-to-graph transformation flow diagram

Input. The points of interest and their connections, hereafter vertices and edges, are of a predefined size, shape and color. For example, each vertex can be a 23×23 pixel square of solid black color and each edge can be straight black line of 2-point width.

Image Loading. The input colored image is transformed into a 3-dimensional matrix where element value (x, y, z) denotes the luminosity of the pixel in position (x, y) for each of the three color channels Red, Green, Blue, having a value between 0 and 255. Black color has a luminosity value 0 for all color channels. Thus, without loss of generality, we can replace the initial 3-dimensional matrix with a 2-dimensional equivalent working matrix corresponding to the green color channel.

Vertex Recognition. In order to detect marked vertices in the input image, our algorithm searches for blocks of zeros of size 23×23 in the working matrix. When such a block is detected, the corresponding vertex and the coordinates of its center, $v_{x,y}$, are stored in the set of vertices V. Then, vertices are sorted from top left to bottom right and numbered.

Edge Recognition. For detecting an edge between two vertices the following assumption is made. The edge originates from the center of a vertex or a point very close to it and terminates at the center of another vertex or, respectively, at a point very close to it. Clarification examples are presented in Fig. 6, where vertex centers are indicated as red dots and edge start and end points are indicated as blue dots.

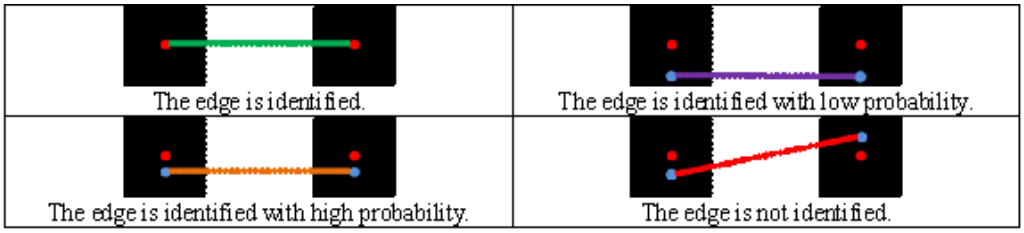

Figure 6. Detection of an edge between two vertices

In order to identify all edges between the vertices, we examine all possible combinations of two overall detected vertices, thus performing $\binom{|V|}{2} = O(|V|^2)$ checks.

It is worth-mentioning that the idea underlying our edge detection algorithm comes from another hot topic of state-of-the-art research with a high impact on human studies as well, namely, finding habitable extrasolar planets orbiting around other stars. The method which has given astronomers almost 80%[1] (so far) of the confirmed extrasolar planets is the transit technique which searches for shadows and is based on the observation of a periodical decrease in the brightness of a star (Fig. 7)[2].

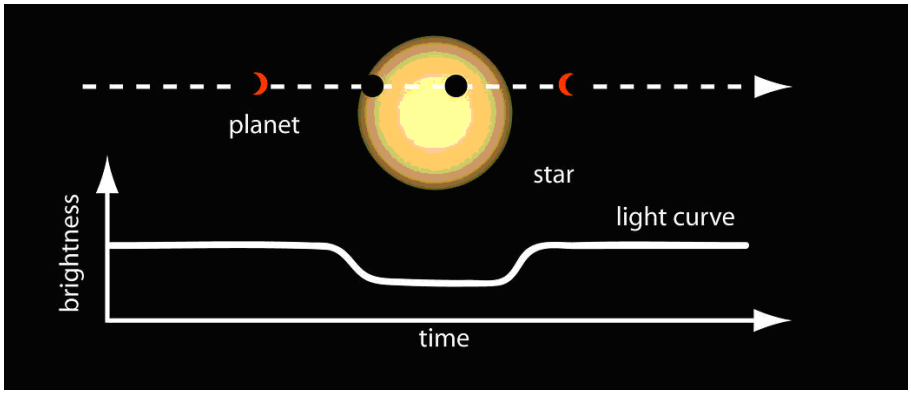

Figure 7. Transit Technique (source: NASA, Kepler Mission)

In the case of two vertices V_1 and V_2 with coordinates $(V_{1,x}, V_{1,y})$ and $(V_{2,x}, V_{2,y})$, we are calculating the sum of luminosity values of the pixels in a search zone that could contain the candidate edge (Fig. 8).

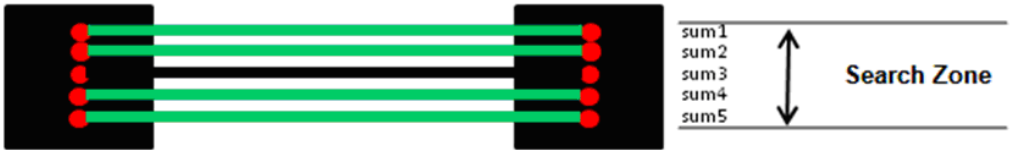

Figure 8. Search Zone for candidate edge

For calculating sum1 we simply add the luminosity values of the pixels that lay on the top green line. We calculate the rest of the sums in the same fashion. In the case of two vertices in horizontal or vertical position, the coordinates for each pixel are easily calculated by keeping coordinate x (or y, respectively) constant and assigning y (or x, respectively) all possible values. When vertices are in diagonal position (Fig. 9), the same idea is applied and the position of each pixel (x, y) is calculated based on the equation $y = a \cdot x + b$, yielding

$$a = \frac{V_{2,y} - V_{1,y}}{V_{2,x} - V_{1,x}} \text{ and } b = V_{1,y} - a \cdot V_{1,x} \quad (1)$$

[1] https://exoplanets.nasa.gov/interactable/11/

[2] https://www.nasa.gov/mission_pages/kepler/multimedia/images/transit-light-curve.html

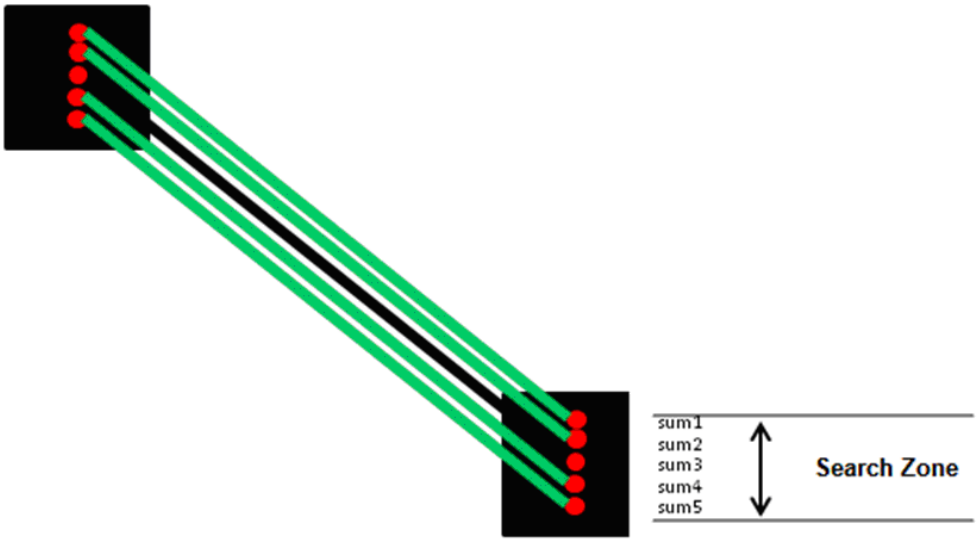

Figure 9. Diagonal Search Zone

Since the sample pixels for calculating the corresponding sums are taken by increasing or decreasing the row in our working matrix, i.e., the x coordinate, the accuracy of the search depends on the population of the samples. For small slope values, candidate edges produce very few sample pixels (Fig. 10)

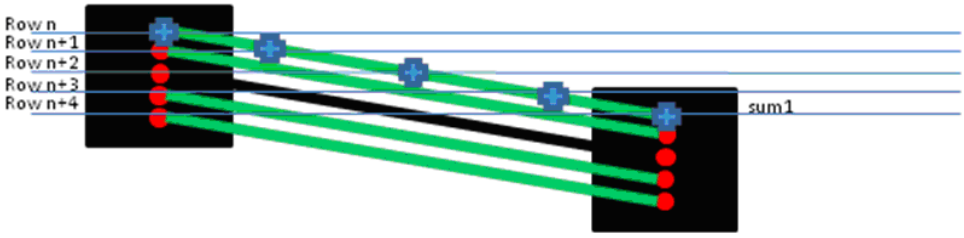

Figure 10. A candidate edge with small slope value

In such a case, we invert the axis and instead of calculating the y coordinate by using $y = a \cdot x + b$, we calculate the x coordinate using $x = \frac{y-b}{a}$, where a and b are the values calculated by equation (1). This results in more populated sample pixels, thus allowing for a more accurate approximation of the candidate edge (Fig. 11).

Having calculated the sums of the luminosity values in the search zone, it is then straightforward to confirm the existence of an edge between two vertices by observing a significant decrease in the luminosity values (e.g., luminosity value below a pre-defined threshold). See, for example, the two diagrams shown in Fig. 12 for candidate edges between vertices 1 and 4 (which are indeed connected by a edge) and vertices 1 and 5 (which are not connected by an edge) from Fig. 2.

In the left part of Fig. 12, regarding vertices 1 and 4, a clear pattern emerges similar to the one of Fig. 7. Since there are sample pixels that lay on a line given by an equation $y = a_{edge} \cdot x + b_{edge}$

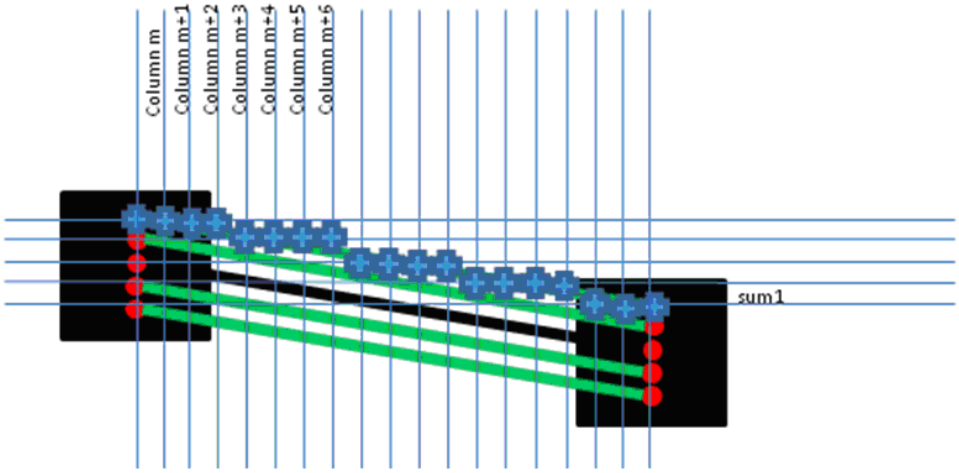

Figure 11. More accurate approximation of a candidate edge

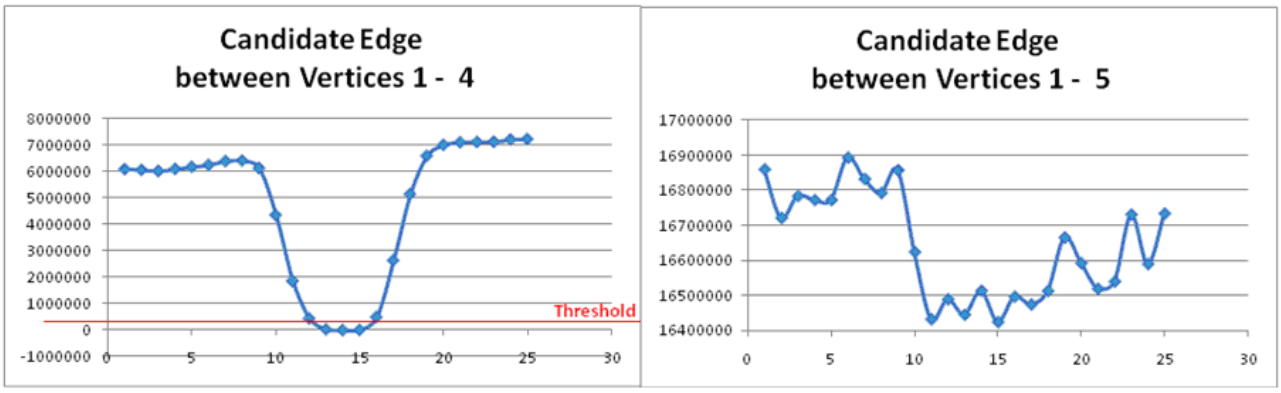

Figure 12. Sum Luminosities of sample pixels in search zones

with a total luminosity approaching zero, and this behavior is observed in all three color channels (implying black color for all pixels along this line), there is a clear indication for the existence of an edge between these two vertices. On the contrary no edge is identified between vertices 1 and 5. Following this process for all pairs of vertices, we identify all edges in the graph.

Edge-Weight Assignment. Identified vertices are tagged with a number in yellow background (Fig. 2). Each identified edge is assigned a weight denoting the construction cost of the specific edge. Edges are then tagged with a number in red background indicating the edge weight (Fig. 3).

Output. An adjacency matrix is created, where rows and columns correspond to graph vertices. A value "0" in the element (V_1, V_2) in the adjacency matrix indicates that no edge exists between those two vertices; a positive value "p" indicates that an edge exists between these two vertices and it has a weight equal to p. In Fig. 13, the adjacency matrix for the input image of Fig. 3 is depicted.

0	2	4	1	0	0	0	0	0	0
2	0	0	6	0	5	0	0	0	0
4	0	0	0	0	1	0	0	0	0
1	6	0	0	2	8	7	0	0	0
0	0	0	2	0	0	2	0	0	0
0	5	1	8	0	0	0	3	0	0
0	0	0	7	2	0	0	4	2	0
0	0	0	0	0	3	4	0	1	3
0	0	0	0	0	0	2	1	0	5
0	0	0	0	0	0	0	3	5	0

Figure 13. Adjacency matrix with weights for the input image of Fig. 3

Accuracy. The accuracy of the image-to-graph transformation algorithm is defined in terms of the correctness of the produced graph. This means that image to graph transformation is accurate if all vertices and edges in the input image also exist in the produced graph.

Indeed, there is a correspondence between vertices in the input image and vertices in the produced graph. Recognition is performed by finding a matrix within a matrix. In particular, the algorithm searches for blocks of zeros in the working matrix which corresponds to an input image. Since vertices are discrete and there are always pixels with non-zero value between them, the algorithm always recognizes all vertices, even those with minimum distance between them. More precisely, the size of the vertex search block is set as a parameter while each vertex in the input image has a predefined size, i.e., 27×27 pixels. However, pixels at the boundary of a vertex may not have 0 luminosity (due to the presence of the black color) but a low luminosity value instead (e.g., 50). So, using a vertex search block of size 27×27 may cause loss of vertices. For avoiding this sort of loss, we use a search block of size slightly smaller than the vertex size (i.e., 23×23). In this way, each vertex can be successfully recognized more than once resulting in (almost similar) corresponding recognition instances of very small mutual deviation (see Fig. 14). For keeping execution time as low as possible, we take into account only the first successful recognition instance

for each vertex, ignoring all subsequent ones, and define the vertex center, $C_{r_{x,y}}$, as the center of this first successful recognition instance.

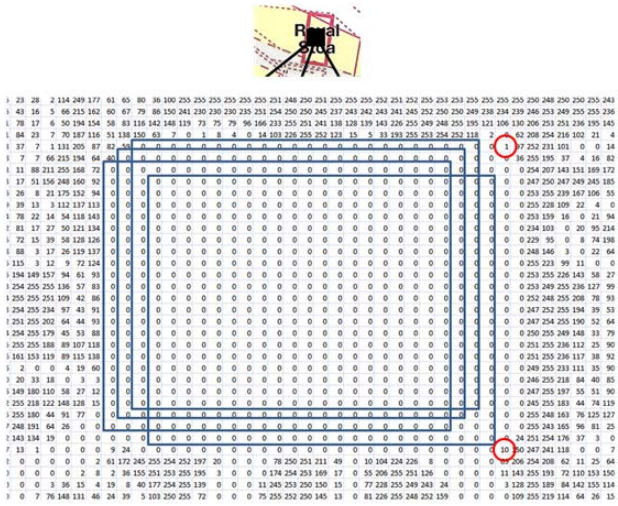

Figure 14. Each vertex can be successfully recognized more than once; only the first successful recognition instance for each vertex is taken into account.

Also, there is a correspondence between edges in the input image and edges in the produced graph. More precisely, horizontal and vertical edges in the input image are immediately recognized due to steep (i.e., 100%) decrease in luminosity. Due to the digital format of the image, diagonal edges are not straight lines but a very good approximation of them resulting from the use of neighboring pixels. We apply a higher tolerance (i.e., 80%) in the decrease of sums of luminosity, and calculate a close-to-real estimation of a candidate diagonal edge. In this way, our algorithm correctly identifies all edges in the input instance. Furthermore, the number of edges in the produced graph never exceeds the number of edges in the input image, since every pair of vertices is examined only once returning a yes/no result regarding the existence of an edge between them. When an edge between two vertices is recognized, then it is "virtually" removed from the input image (by assigning a large luminosity value, e.g., 255, to all pixels of the recognized edge). This approach guarantees that an extra, non-existing edge (A,C) will never be inaccurately counted, even when edges (A,B) and (B,C) exist in the input image and all three vertices A, B and C are aligned as a straight line.

In addition, experimental evaluation on several (roughly, 50) input images from real, easy-to-verify places showed 100% correctness in the graph production.

Another interesting but not crucial accuracy check concerns the exact position of recognized graph elements (i.e., vertices and edges) in comparison to the actual position of these elements in the input image. However, accuracy in the exact position of recognized graph elements can be important only for drawing a realistic MST.

In particular, the accuracy of our image recognition algorithm in terms of the position of the recognized vertices is defined in terms of the distance (expressed in pixels) between a recognized vertex and its real world equivalent in the input image. More precisely, for each vertex in the input image, we denote by $C_{R_{x,y}}$ the actual center of this vertex and by $C_{r_{x,y}}$ the center of the

corresponding vertex recognized by our technique. Then, the differences $Err_{rR_x} = C_{r_x} - C_{R_x}$ and $Err_{rR_y} = C_{r_y} - C_{R_y}$ indicate the error in the recognition result. The lower the average error values, the higher the obtained accuracy. Experimental evaluation on several input instances confirms a 99.9% accuracy rate. For instance, Fig. 15 presents the error in the horizontal and vertical coordinates between calculated and actual centers in a graph of 10 vertices and 16 edges resulted from a sample input image of 1127 rows and 1252 columns.

Calculated Center		Actual Center		Vertical Error	Horizontal Error
y coordinate	x coordinate	y coordinate	x coordinate	y coordinate	x coordinate
188	428	189	427.5	0.08873%	0.03994%
317	381	318	380.5	0.08873%	0.03994%
410	147	411.5	146	0.13310%	0.07987%
434	569	435	568	0.08873%	0.07987%
563	1026	564	1025	0.08873%	0.07987%
575	286	576	286.5	0.08873%	0.03994%
586	698	587.5	697	0.13310%	0.07987%
692	323	693	322	0.08873%	0.07987%
797	604	797.5	603	0.04437%	0.07987%
914	475	915.5	474.5	0.13310%	0.03994%
		Average Error		0.09760%	0.05591%

Figure 15. Accuracy of recognition of 10 vertices and 16 edges in a 1127×1252 input image.

Complexity. Vertex recognition is performed by filtering the matrix which corresponds to the input image for fixed-sized blocks of black pixels. Such blocks indicate the existence of a vertex in the input image. This is obtained by sliding a search block (i.e., a 23×23 matrix of zeros) throughout the matrix corresponding to the input image. Using appropriate functions of MATLAB, the task of vertex location is reduced to a search in a line vector, which can be performed quickly via simple arithmetic calculations involving small integers (i.e., coordinates). However, this process returns multiple instances of the same vertex, which are then eliminated by simply sorting the coordinates of the detected vertices and ignoring redundant instances of the same vertex. Edge recognition is performed by checking all pairs of located vertices thus requiring $\binom{|V|}{2} = O(|V|^2)$ checks. So, the overall complexity of the image-to-graph transformation algorithm remains polynomial in the number of vertices existing in the input image. Experimental evaluation on several (roughly, 50) real input images containing up to 30 vertices also confirm the efficiency of image-to-graph transformation.

3.2. Prim's algorithm implementation

The flow diagram of our implementation of Prim's algorithm is shown in Fig. 16.

Input. the adjacency matrix of the input weighted graph returned by the image-to-graph transformation process (see Fig. 13). MST computation: We first augment the the adjacency matrix provided as input by adding a row and a column; cell values correspond to vertex labels (Fig. 17).

We start with vertex 1 (column 2) and add it to the growing MST. For each vertex already in the MST (indicated by its corresponding column), we find an incident edge of minimum cost (i.e.,

Prim's algorithm implementation

- **Input:** Adjacency matrix for the resulting weighted graph
- **Process**
  1. MST computation
  2. Display result
- **Output:** Image (.png) with MST overlaying the initial graph

Figure 16. Implementation of Prim's algorithm

-1	1	2	3	4	5	6	7	8	9	10
1	0	2	4	1	0	0	0	0	0	0
2	2	0	0	6	0	5	0	0	0	0
3	4	0	0	0	0	1	0	0	0	0
4	1	6	0	0	2	8	7	0	0	0
5	0	0	0	2	0	0	2	0	0	0
6	0	5	1	8	0	0	0	3	0	0
7	0	0	0	7	2	0	0	4	2	0
8	0	0	0	0	0	3	4	0	1	3
9	0	0	0	0	0	0	2	1	0	5
10	0	0	0	0	0	0	0	3	5	0

Figure 17. Augmented adjacency matrix for the MST computation

we find the minimum positive value in that column) that connects a vertex currently not included in the growing MST. We then insert them both (edge and vertex) into the MST. We exclude row values corresponding to vertices already included in the MST. We repeat this process for $|V| - 1$ times. In this way, we produce the MST adjacency matrix which contains weight values only for edges included in the MST. Fig. 18 presents the MST adjacency matrix produced for the input image of Fig.3.

Display result. For each MST edge, we overlay a red line on the image, setting the appropriate value to the corresponding pixels based on the equation $y = a_{edge} \cdot x + b_{edge}$ (or the equation $x = \frac{y - b_{edge}}{a_{edge}}$ depending on the value of the slope) for calculating pixel coordinates. We display the image with the identified vertices and the edges belonging to the MST colored in red (Fig. 4).

Output. We, finally, save the generated image as a *.png* file with a filename resulting from that of the initial input image concatenated with the suffix "_MST".

4. Concluding Remarks

Motivated by the need for making heritage sites more accessible and providing equal access for all, we designed and implemented an application which automatically computes efficient construction plans (for escalator or low-emission hot spots) for connecting all points of interest in cultural sites. Providing as input an image or a simple photograph of the site in question, the administration of cultural sites can quickly obtain an efficient construction plan which can be used to provide

0	2	0	1	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
1	0	0	0	2	0	0	0	0	0
0	0	0	2	0	0	2	0	0	0
0	0	1	0	0	0	0	3	0	0
0	0	0	0	2	0	0	0	2	0
0	0	0	0	0	3	0	0	1	3
0	0	0	0	0	0	2	1	0	0
0	0	0	0	0	0	0	3	0	0

Figure 18. Generated MST adjacency matrix for the input image of Fig. 3

an escalator construction plan of minimum total length. Furthermore, the plan generated by our application also suggests an efficient plan for placing low-emission routers for wireless guiding services and transmission of cultural information to visitors.

In the context of our future plans, we intend to refine our method for image recognition while preserving its simplicity in comparison to particularly more involved relevant commercial products. Also, we plan to include an image pre-processing phase in order to determine the density of the input graph based on vertex distance in order to refine our edge detection technique. In addition, we plan to extend our algorithm to work also for input graphs which are not connected. In such cases, we wish to compute minimum spanning trees for each connected component of the input graph and obtain a minimum spanning forest. Furthermore, we plan to investigate how our application can be appropriately upgraded for autonomous execution and use on mobile devices (e.g., tablets).

Looking at the bigger picture, our approach highlights how graph theory, relevant algorithmic solutions and a simple programming environment can be fruitfully blended to provide a practical solution to the real and significant requirement for global physical access to cultural heritage sites so that all people are supported to enjoy their right to freely and equally participate in the cultural life of the community.

References

References

[1] L. An, Q.S. Xiang and S. Chavez, A fast implementation of the minimum spanning tree method for phase unwrapping, *IEEE Transactions on Medical Imaging* **19** (8) (2000), 805–808.

[2] D.A. Bader and G. Cong, Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs, *Journal of Parallel and Distributed Computing* **66** (11) (2006), 1366–1378.

[3] O. Borůvka, O jistém problému minimálním [About a certain minimal problem], *Práce mor. přírodověd. spol. v Brně III* (in Czech and German) **3** (1926), 37–58.

[4] O. Borůvka, Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí (Contribution to the solution of a problem of economical construction of electrical networks), *Elektronický Obzor* (in Czech) **15** (1926), 153–154.

[5] B. Chazelle, A minimum spanning tree algorithm with inverse-Ackermann type complexity, *Journal of the ACM* **47** (6) (2000), 1028–1047.

[6] C. Chen and S. Morris, Visualizing evolving networks: minimum spanning trees versus Pathfinder networks, *In Proceedings of the IEEE Symposium on Information Visualization* (2003), 67–74.

[7] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, Introduction to Algorithms (2nd ed.), *MIT Press and McGraw-Hill* (2001) ISBN 0-262-03293-7.

[8] M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *Journal of the ACM* **34** (3) (1987), 596–615.

[9] R.L. Graham and P. Hell, On the history of the Minimum Spanning Tree problem, *Annals of the History of Computing* **7** (1) (1985), 43–57.

[10] E. Horowitz and S. Sahni, Fundamentals of Computer Algorithms, *Computer Science Press* (1978)

[11] V. Jarník, O jistém problému minimálním (About a certain minimal problem), *Práce Moravské Přírodovědecké Společnosti* (in Czech) **6** (1930), 57–63.

[12] D.R. Karger, P.N. Klein and R.E. Tarjan, A randomized linear-time algorithm to find minimum spanning trees, *Journal of the ACM* **42** (2) (1995), 321–328.

[13] J. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, *In Proceedings of the American Mathematical Society* **7** (1956), 48–50.

[14] M. Mareš, The saga of minimum spanning trees, *Computer Science Review* **2** (3) (2008), 165–221.

[15] S. Meguerdichian, F. Koushanfar, M. Potkonjak and M. Srivastava, Coverage problems in wireless ad-hoc sensor networks. *In Proceedings of the INFOCOM '01*, IEEE Press, (2001), 1380–1387.

[16] C. Moler, The Origins of MATLAB, *Mathworks* (2004)

[17] J. Nešetřil, E. Milková and H. Nešetřilová, Otakar Borůvka on minimum spanning tree problem: translation of both the 1926 papers, comments, history, *Discrete Mathematics* **233** (1-3) (2001), 3–36.

[18] J. Nešetřil and H. Nešetřilová, The Origins of Minimal Spanning Tree Algorithms - Borůvka and Jarník, *Documenta Mathematica*, **Extra Volume: Optimization Stories** (2012), 127–141.

[19] V. Olman, D. Xu and Y. Xu, Identification of regulatory binding sites using minimum spanning trees, *In Proceedings of the 8th Pacific Symposium on Biocomputing (PSB 2003)*, World Scientific (2003), 327–338.

[20] E. Papaioannou and T. Karatasos, Automatic MST Generation for Efficient Global Physical Access to Cultural Sites, *In Proceedings of the 4th International Conference on Education, Social Sciences and Humanities (SOCIOINT 17)* (2017), in press.

[21] S. Pettie, Finding minimum spanning trees in $O(m\alpha(m,n))O(m\alpha(m,n))$ time, *Tech Report TR99-23, University of Texas at Austin* (1999)

[22] S. Pettie and V. Ramachandran, An optimal minimum spanning tree algorithm, *Journal of the ACM* **49** (1) (2002), pp. 16–34.

[23] R.C. Prim, Shortest connection networks and some generalizations, *Bell System Technical Journal* **36** (6) (1957), 1389–1401.

[24] F.S. Roberts, Graph theory and its applications to the problems of society, *CBMS-NSF Monograph 29* (1978), SIAM Publications.

[25] J. Xu, Theory and application of graphs, *Springer Science and Business Media, LLC* (2003). ISBN 9778-1-4613-4670-8

[26] Access: Improving the accessibility of historic buildings and places, Department of Arts, Heritage and the Gaeltacht Affairs and National Disbility Authority, *Government Publications Sales Office, Dublin, Ireland* (2011) (http://nda.ie/Publications/Environment-Housing/Environment-Publications/Access-Improving-the-accessibility-of-historic-buildings-and-places.html)

[27] United Nations, Universal Declaration of Human Rights (1948) (http://www.un.org/en/universal-declaration-human-rights/)