# AUTOMATIC MST GENERATION FOR EFFICIENT GLOBAL PHYSICAL ACCESS TO CULTURAL SITES

**Thodoris Karatasos[1], Evi Papaioannou[2]***

[1]University of Patras, GREECE, tkaratasos@upatras.gr
[2]University of Patras and CTI "Diophantus", GREECE, papaioan@ceid.upatras.gr
*Corresponding Author

## Abstract

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. 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. 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, i.e., there is a path between every pair of locations in it, 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, in short) problem.

In graph theory, a graph, usually denoted as G=(V,E), is composed of a set of vertices, V, and a set of edges, E, where edges connect pairs of (usually distinct) vertices. A predefined function indicates which pairs of vertices are connected by an edge; so, the existence of an edge implies a physical or logical relation between its endpoints, i.e., the vertices it connects. If edges are simple lines without a particular direction then the corresponding graph is called undirected. Edges can also have a number associated with them which is called weight or cost of the edge resulting in edge-weigthed graphs. When there is a path, i.e., a sequence of consecutive edges, between every pair of vertices in a graph, then this graph is said to be connected. A graph is a tree when it is undirected, connected and does not include simple circuits, i.e., paths with no repeated edges that start and end at the same vertex. In graph-theoretic terms, the MST problem can be stated as follows. Given a connected, undirected, edge-weighted graph G=(V,E,w), the objective of the MST problem is to compute a tree of minimum weight containing all vertices of V.

In the context of efficient global physical access to cultural sites, corresponding site locations to vertices and adding an edge between two vertices as long as there is a physical connection between the corresponding site locations makes computing an efficient escalator construction plan equivalent to an instance of the Minimum Spanning Tree problem in the underlying graph. The Minimum Spanning Tree problem can be solved quickly (i.e., in polynomial time in the size of the input) using well-known algorithms from the relevant literature. Following these lines, 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 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. Looking at the bigger picture, our approach clearly highlights how graph theory and relevant algorithmic solutions can be used to provide efficient practical solutions to a wide range of real world problems.

**Keywords**: Applications of graph theory to problems of society, MST, image recognition, matlab, global physical access, culture, tourism.

## 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” (United Nations, 1948, Article 27 (1)). 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.

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. 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 (Access, 2011).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 (Roberts, 1978).

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, i.e., there is a path between every pair of locations in it, 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, in short) problem.Then, exploiting existing efficient algorithmic techniques for the MST problem, we designed and implemented, using the MATLAB programming environement, an application which automatically computes efficient construction plans 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. 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 Minimum Spanning Tree 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 MINIMUM SPANNING TREE PROBLEM

The Minimum Spanning Tree problem is one of the typical and best-studied 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 (Horowitz, Sahni, 1978, Graham, Hell, 1985, Cormen, Leiserson, Rivest, Stein, 2001). 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 minimized.

More formally, a graph, usually denoted as G=(V,E), is composed of a set of vertices, V, and a set of edges, E, where edges are 2-element subsets of V representing a connection between two vertices. A predefined function indicates which pairs of vertices are connected by an edge; so, the existence of an edge implies a physical or logical relation between its endpoints, i.e., the vertices it connects. Edges can be directed or undirected. If edges are simple lines without a particular direction then they are undirected and the corresponding graph is called undirected. Edges can also have a number associated with them which is called as a weight or cost. When there is a path, i.e., a sequence of consecutive edges, between every pair of vertices in a graph, then this graph is said to be connected. A graph is a tree when it is undirected, connected and does not include simple circuits, i.e., paths with no repeated edges that start and end at the same vertex. A spanning tree of a simple graph G = (V,E) is a subgraph of G that is a tree containing every vertex of G, i.e., spanning all vertices of G. An edge-weighted graph is a graph where each edge is assigned a weight or cost. In edge-weighted graphs, generated spanning trees have a weight which equals the sum of the weights of their edges. Then, formally, the MST problem can be stated as follows. Given an undirected graph G=(V,E), where V denotes the set of vertices with |V|=n and E denotes the set of edges with |E|=m, and a real number w(e) for each edge $e \in E$ which is called the weight of edge e, the MST problem is formally defined as finding a spanning tree T* on G, such that $w(T^*)=\min_T \Sigma_{e \in T} w(e)$ is the minimum calculated over all possible spanning trees of G. A more intuitive definition can be found in (Horowitz, Sahni, 1978).

Despite that it is quite intuitive to associate edge weights with distances, edge weights can essentialy reflect any arbitrary property, like for example time, financial cost, energy units, contextual/conceptual correlation, etc. This further implies that edge weights can be zero or negative raising alternative definitions for the minimality requirement. In the general case, edge weights are different; in the special case when all edge weights are equal, there may exist multiple spanning trees for a given connected simple graph. Furthermore, if the graph is not connected, minimum spanning trees for each of its connected components are computed generating a minimum spanning forest.

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 essentialy 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 (Graham, Hell, 1985, An, Xiang, Chavez, 2000, Olman, Xu, Xu, 2003, Chen, Morris, 2003). Furthermore, the MST problem often occurs as a subproblem in the context of other problems. For example, Minimum Spanning Tree 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 (Graham, Hell, 1985).

## 2.1 General solution and classical algorithms

Traces of the Minimum Spanning Tree problem date back to 1926 when Borůvka first studied primarily an Euclidean version of the problem related to planning of electrical transmission lines (Borůvka, 1926(a)), and suggested an efficient algorithm for the general version of the problem (Borůvka, 1926(b)). Borůvka's work was further extended by Jarník (Jarník, 1930), 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 $O(m\beta(m,n))O(m\beta(m,n))$ time algorithm by Fredman and Tarjan (Fredman, Tarjan, 1987), over algorithms with inverse-Ackermann-type complexity by Chazelle (Chazelle, 2000) and Pettie (Pettie, 1999), to an algorithm by Pettie and Ramachandran (Pettie, Ramachandran, 2002) whose time complexity is provably optimal. Frequently, the most important ingredients were advances in data structures used to represent the graph (Mareš, 2008).

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.

While there is a long list of algorithms for the MST problem, we provide a detailed description for three classical MST construction algorithms, namely Borůvka's algorithm (Borůvka, 1926(b)), Kruskal's algorithm (Kruskal, 1956) and Prim's algorihtm (Prim, 1957). Borůvka's algorithm has been the basis for several improved algorithms for the MST problem. The algorithms of Kruskal and Prim 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.

**Borůvka's algorithm** (Borůvka, 1926(a), Nĕsetřil, Nĕsetřilová, 2012): 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. Then, it joins these groupings in a similar manner until a tree spanning all graph vertices is generated. 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, since it requires O(log|V|) iterations while each iteration needs |E| steps. However, improved versions have been suggested (Karger, Klein, Tarjan, 1995, Chazelle, 2000).

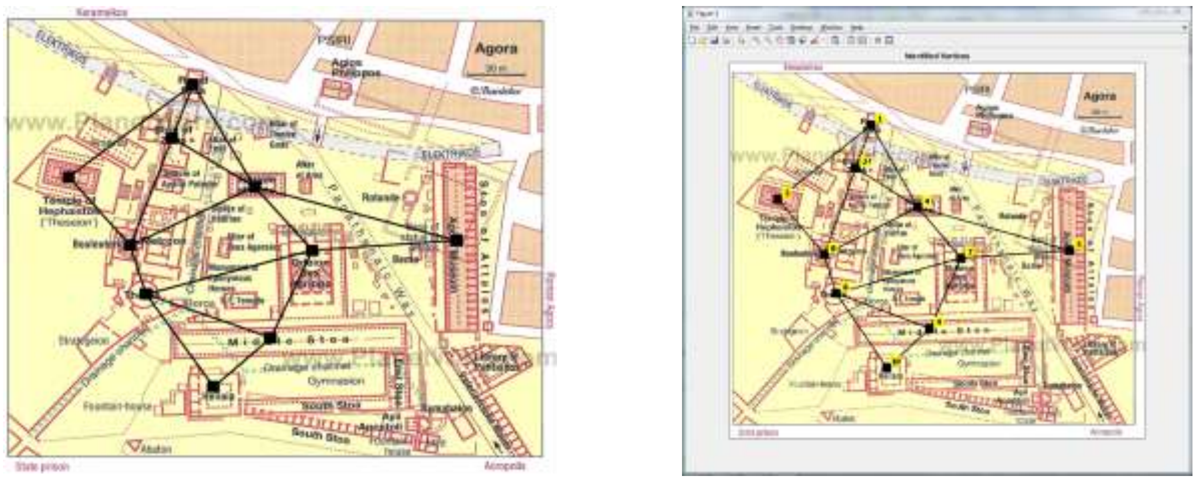
**Kruskal's algorithm** (Kruskal, 1956): the algorithm starts with an edge in the graph with minimum weight and builds the spanning tree by successively adding edges one by one into a growing spanning tree. It processes the edges in order of their weight values, from smallest to largest, including into the growing MST an edge as long as it 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.

**Prim's algorithm** (Jarník, 1930, Prim, 1957): 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 candidates nearest vertices, ties can be broken arbitrarily. The algorithm terminates when all vertices of the graph have been included in the spanning tree. Since each vertex is considred only once, the time complexity of the Prim's Algorithm is O((V+E)logV). However, improved versions have been suggested (Bader, Cong, 2006).

## 3 OUR APPLICATION FOR AUTOMATIC MST GENERATION FOR CULTURAL SITES

We implemented our application using the MATLAB programming environment (Moler, 2004). 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.

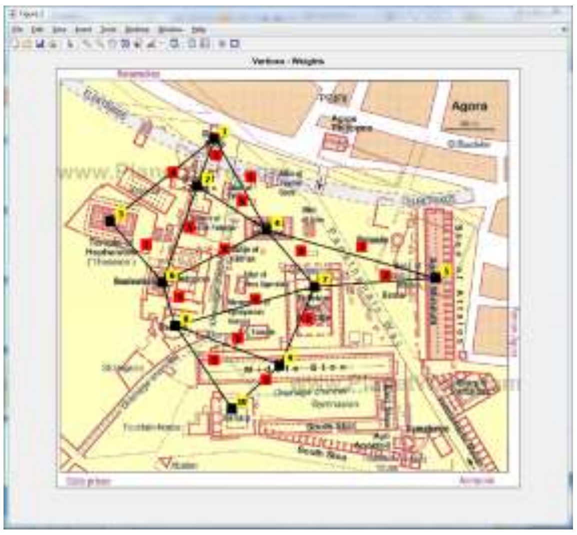
Our application receives as input an image file of a cultural site where points of interest have been appropriately marked and connected (Fig. 1(a)). The image is then processed and transformed into a working graph, where points of interest correspond to vertices and there is an edge between two vertices if the corresponding points of interest are actually connected by means of a physical link. In the resulting graph (Fig. 1(b)), vertices indicated with a yellow number have been identified and stored together with the edges connecting them in the graph.



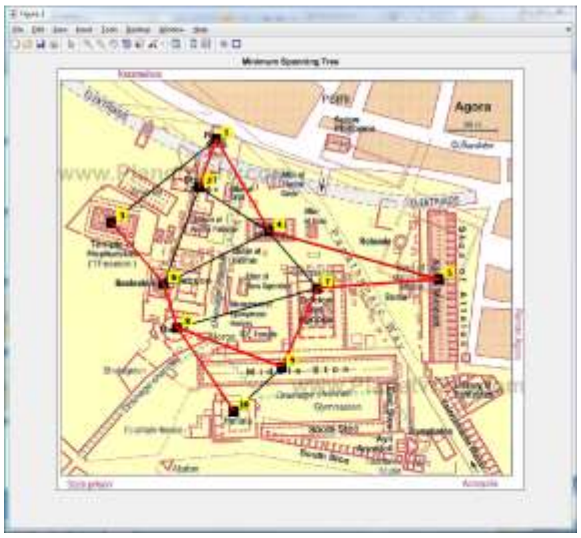
(a) (b)

Fig. 1: (a) A sample input image for the archaeological site of Agora in Ancient Athens, Greece and (b) the resulting graph.

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. 2(a)). 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. 2(b)).



(a)



(b)

Fig. 2: (a) The weighted graph and (b) the computed Minimum Spanning Tree.

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.

### *3.1.1 Automatic image-to-graph transformation*

The flow diagram of the automatic image-to-graph transformation is shown in Table 1.

Table 1: Automatic image-to-graph transformation Flow Diagram

| 1. Automatic image-to-graph transformation | |
|---|---|
| Input | Image (.png) with marked points of interest as vertices and connections as edges between them |
| Process | 1.1 Image Loading<br>1.2 Vertex Recognition<br>1.3 Edge Recognition<br>1.4 Edge-Weight Assignment |
| Output | Adjacency Matrix for the resulting weighted graph |

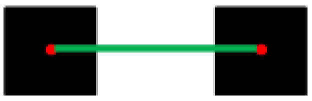
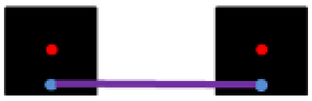
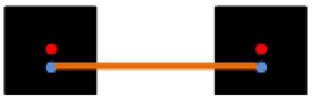
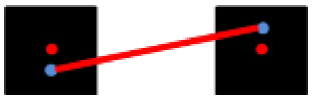
**Input**: The points of interest and their connections, hereafter vertices and edges, are of a predefined size, shape and color. For example, each vertex can be a 23x23 pixel square of solid black color and each edge can be straight black line of two-point width.

**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 23x23 in the working matrix. When such a block is detected, the corresponding vertex and the coordinates of its centre, $v_{x,y}$' 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 centre of a vertex or a point very close to it and terminates at the centre of another vertex or, resectively, at a point very close to it. Clarification examples are presented in Table 2, where vertex centres are indicated as red dots and edge start and end points are indicated as blue dots.

Table 2: Detection of an edge between two vertices.

| | |
|---|---|
|  |  |
| The edge is identified. | The edge is identified with low probability. |
|  |  |
| The edge is identified with high probability. | The edge is not identified. |

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

It is worth-mentioning that the idea underlying our edge detection algorithms 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 is based on the observation of a periodical decrease in the brightness of a star (Fig. 3).

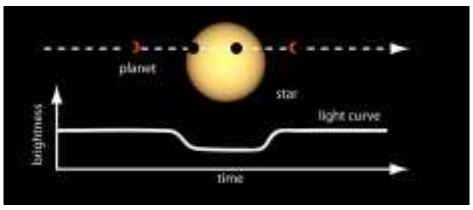


Fig. 3: Transit Technique (source: NASA, Kepler Mission)[2]

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 luminosities of the pixels in a search zone that could contain the candidate edge (Fig. 4).

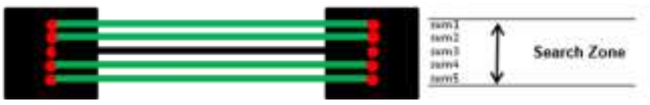


Fig. 4: Search Zone for candidate edge

For calculating sum1 we simply add the luminosities 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, the same idea is applied and the position of each pixel (x,y) is calculated based on the equation y = a*x+b, yielding

$$a = (V_{2,y} – V_{1,y})/(V_{2,x} – V_{1,x}) \text{ and } b = V_{1,y} – a*V_{1,x} \text{ (Eq. 1) (Fig. 5)}$$

[1] https://exoplanets.nasa.gov/interactable/11/
[2] https://www.nasa.gov/mission_pages/kepler/multimedia/images/transit-light-curve.html

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. 6)

In such a case, we invert the axis and instead of calculating the y coordinate by using y=a*x+b, we calculate the x coordinate using x = (y-b)/a, where a and b are the values calculated by equation (Eq. 1). This results in more populated sample pixels, thus allowing for a more accurate approximation of the candidate edge (Fig. 7).

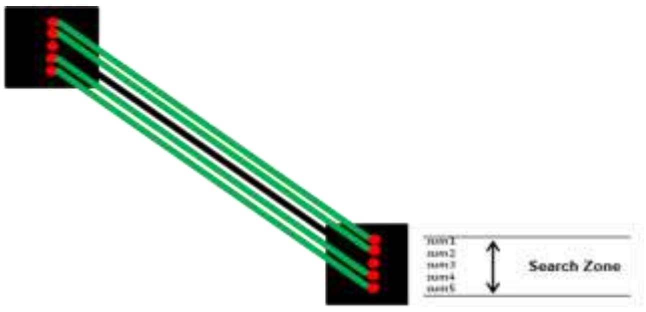


Fig. 5: Diagonal Search Zone.

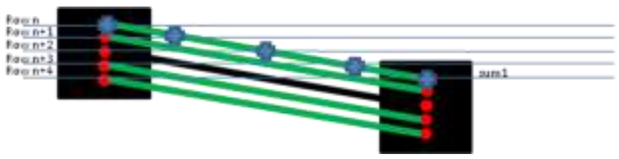


Fig. 6: A candidate edge with small slope value.

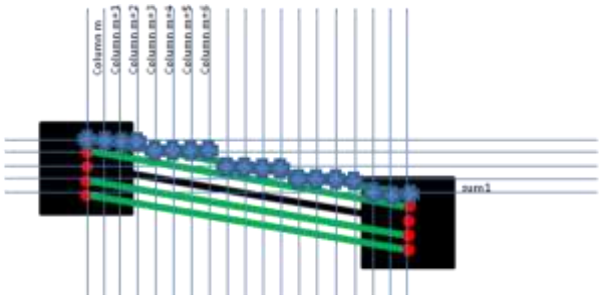


Figure 7: More accurate approximation of a candidate edge.

Having calculated the sums of the luminosities 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 luminosities (e.g., luminosity value below a pre-defined threshold). See, for example, the two diagrams shown in Fig. 8 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. 1(b).

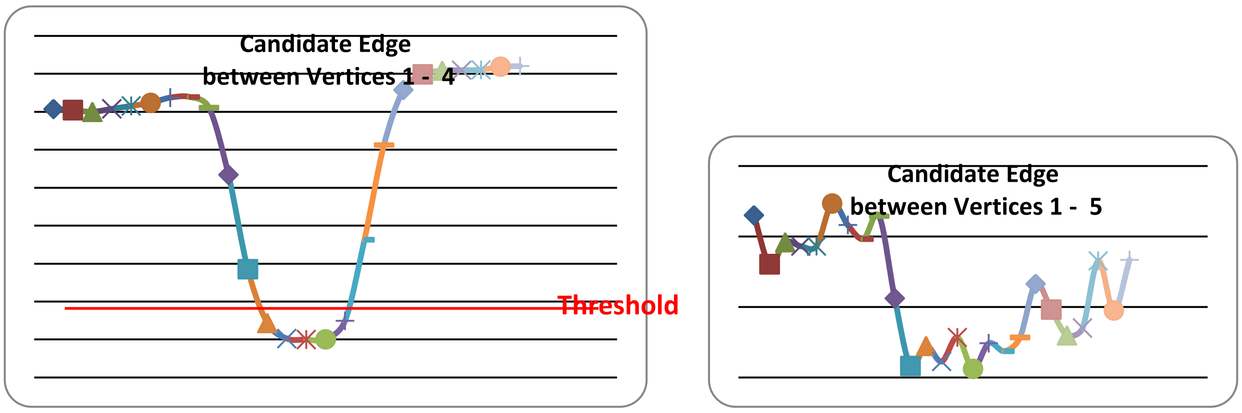


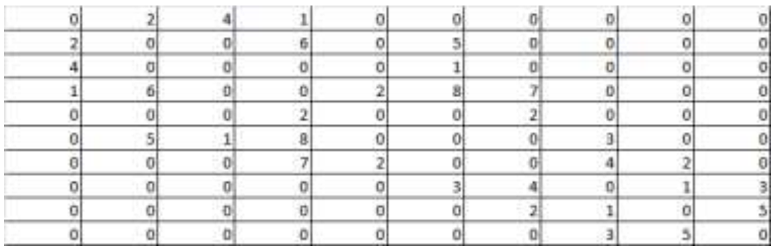
Fig. 8: Sum Luminosities of sample pixels in search zones.

In the left part of Fig. 8, regarding vertices 1 and 4, a clear pattern emerges similar to the one of Fig. 3. Since there are sample pixels that lay on a line given by an equation $y=a_{edge}*x+b_{edge}$ 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. 1(b)). 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. 2(a)).

**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 Table 3, the adjacency matrix for the input image of Fig. 2(a) is depicted.

Table 3: Adjacency matrix with weights for the input image of Fig. 2(a).



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 |

## *3.1.2 Implementation of Prim's algorithm*

The flow diagram of our implementation of Prim's algorithm is shown in Table 4.

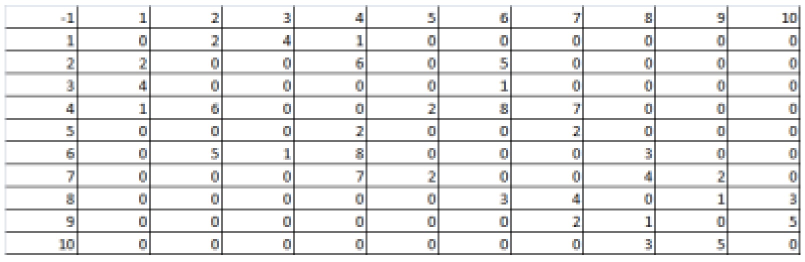
Table 4: Implementation of Prim's algorithm.

| 2. Prim's algorithm | |
|---|---|
| Input | Adjacency matrix for the resulting weighted graph |
| Process | 2.1 MST computation<br>2.2 Display result |
| Output | Image (.png) with MST overlaying the initial graph |

**Input**: the adjacency matrix of the input weighted graph returned by the image-to-graph transformation process (see Table 3).

**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 (Table 5).

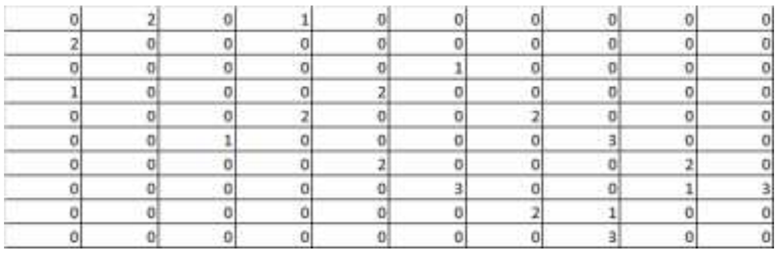
Table 5: Augmented adjacency matrix for the MST compuation.



| -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 |

Staring with vertex 1 (column 2), we 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., 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. See for example Table 6.

Table 6: Produced MST adjacency matrix for the imput image of Fig. 3.



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 |

**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 $y=a_{edge}*x+b_{edge}$ (or the equation $x=(y-b_{edge})/a_{edge}$ 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. 3(b)).

**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 CONCLUSION AND FUTURE PLANS

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 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. In addition, we plan to refine 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.

## REFERENCE LIST

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

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

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

Borůvka, O. (1926(b)). 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), vol. 15, pp. 153–154.

Chazelle, B. (2000). A minimum spanning tree algorithm with inverse-Ackermann type complexity. Journal of the ACM, vol. 47, issue 6, pp. 1028-1047.

Chen, C., Morris, S. (2003). Visualizing evolving networks: minimum spanning trees versus Pathfinder networks. In Proceedings of the IEEE Symposium on Information Visualization, pp. 67 - 74.

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

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

Graham, R.L. and Hell, P. (1985). On the history of the Minimum Spanning Tree problem. Annals of the History of Computing, vol. 7, no 1, pp. 43-57.

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

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

Karger, D. R., Klein, P. N., Tarjan, R. E. (1995). A randomized linear-time algorithm to find minimum spanning trees. Journal of the ACM, vol. 42, issue 2, pp. 321-328.

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

Mareš, M. (2008). The saga of minimum spanning trees. Computer Science Review, Vol. 2, Issue 3, pp. 165–221.

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

Moler, C. (2004). The Origins of MATLAB. Mathworks.

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

Něsetřil, J. and Něsetřilová, H. (2012). The Origins of Minimal Spanning Tree Algorithms - Borůvka and Jarník. Documenta Mathematica, Extra Volume: Optimization Stories, pp. 127–141.

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

Pettie, S. (1999). Finding minimum spanning trees in O(mα(m,n))O(mα(m,n)) time. Tech Report TR99-23, University of Texas at Austin.

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

Prim, R. C. (1957). Shortest connection networks and some generalizations. Bell System Technical Journal, vol. 36, issue 6, pp. 1389–1401.

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

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

"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)

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