EAI Endorsed Transactions
on Smart Cities

Research Article EAI.EU

An approach for rapid generation of interactive spider maps for public transport networks

Sara Santos[1,*], Teresa Galvão[1,2], Thiago Sobral[1,2]

[1]Faculty of Engineering of University of Porto, Portugal
[2]INESC TEC, Porto, Portugal

Abstract

A spider map is a type of schematic map that allows one to answer questions like "From where I am, where can I go?", as it provides only the essential information for a given geographical area (hub), from which lines emerge, whilst keeping the geographic context. They are often designed manually for a limited set of locations, thus reducing its widespread adoption. Moreover, spider maps should conform to several design constraints, which turns the automated generation into a complex problem. Optimisation techniques have been applied to this problem, although existing solutions are time costly and require heavy computational power. This paper presents an approach to automatically generate feasible spider maps within a short execution time based on an algorithm that adapts state-of-the-art methods, producing adequate quality maps to be manipulated in interactive media, based on the areas selected by the user. We report the results of a case study for areas in the city of Porto, Portugal.

Received on 02 February 2020; accepted on 26 August 2020; published on 27 August 2020

Keywords: spider maps, schematic maps, public transport, automation.

doi:10.4108/eai.18-8-2020.166007

1. Introduction

Major cities have complex public transport systems that are part of citizens' daily commuting. Such systems ought to be encouraged as an alternative to private transport. Public transport maps provide simplified representations of their corresponding network infrastructure, thus they should be of easy interpretation, aiming to facilitate the user experience and to increase public transport ridership.

Public transport maps are often represented by schematic maps, since they fulfill the need for simple and effective representation of complex networks [1], with indication of the available services and navigation alternatives. Schematic maps are subject to a number of generalisation and simplification processes so as to translate the mental representation of the network, depicting the services and commuting possibilities within the map range. A specific type of schematic map is the *spider map*, which can be used to represent complex areas like bus networks in city centres. For instance, Figure 1 depicts a spider map for the city of Porto, Portugal which depicts the surroundings of St. John's Hospital (Hospital de São João).

Spider maps allow one to answer questions like "From where I am, where can I go?", as they indicate the travel possibilities from a small geographical area. They are useful during the pre-planning state of a trip as it provides better geographical context by eliminating the visual clutter derived from other information available in the map that is not relevant to that area, in contrast to schematic maps. The central element of a spider map is the hub – a rectangular geographic map – that introduces the spatial context from which the schematic lines emerge. The automatic construction of spider maps is a complex problem, as it is subject to a number of design constraints that should be obeyed, e.g. line angles, location of lines emerging from the hub, spatial constraints due to real-world features like rivers, bridges, etc.

Although spider maps are effective for providing passengers with public transport information, some factors impact their widespread adoption in transport

*Corresponding author. Email: up201402814@fe.up.pt
†Co-authors. tgalvao@fe.up.pt, thiago.sobral@fe.up.pt

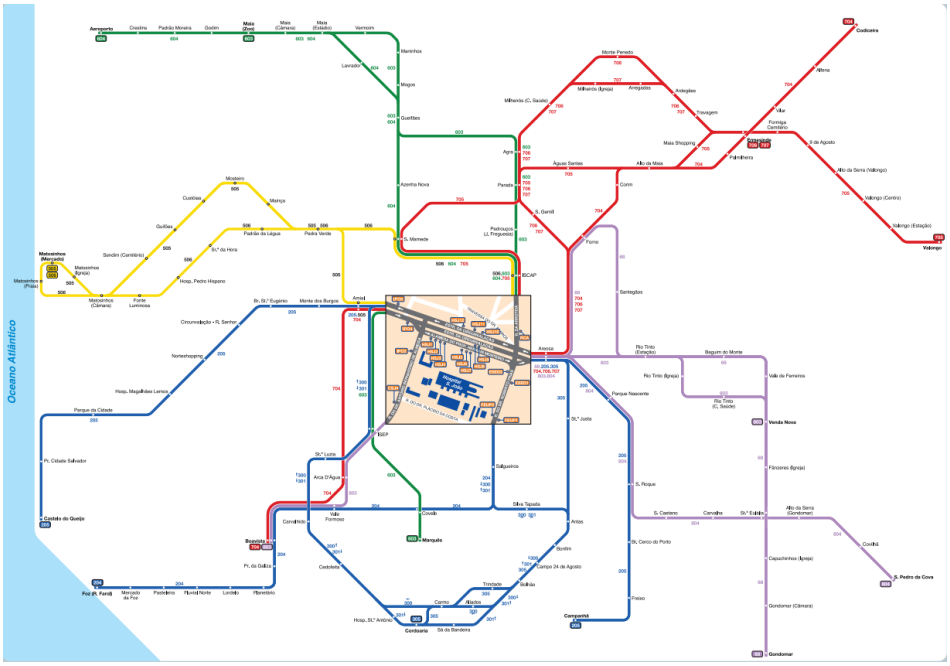


Figure 1. The spider map for the St. John's Hospital (Hospital de São João) transport hub in Porto, Portugal. The map's centre contains a tile image that represents the location of several bus stops (the hub). Several lines emerge from the tile's streets. [2]

networks. Firstly, the generation of such maps is frequently manual, and depends on the expertise of designers. The related studies provide a number of methods and techniques to automate the generation of spider maps, but current solutions based on multi-criteria optimisation algorithms are time-costly and require heavy computing power. Secondly, as an implication, spider maps are created for a few major locations of a city only. We argue that such a limitation undermines the potential of spider maps, as they can be useful to passengers from any location of a transportation network served by various lines.

This paper proposes an approach to generate feasible interactive spider maps within a small timeframe, based on an algorithm that modifies and adapts some of the state of the art techniques. The goal is to tackle the complexity of the problem and present viable solutions with short execution times and using less computational power. Thus, it aims at simplifying the traditional spider map generation process and potentially make an impact on the use of spider maps. Bringing interactive capabilities to spider maps allows citizens to actively explore their transport network and a have map that is tailored to the desired geographic area, whilst introducing new challenges for generation of such maps within an acceptable time frame, especially if they are available for interactive displays like smartphones and tablets, and kiosks placed in stations and streets. Users can leverage the potential of the proposed approach to generate spider maps for virtually any city area, and become aware of not only the nearby stops, but how far can he go from that area by boarding one of the available services.

This paper extends the work described in [3] with the following contributions: a more comprehensive description of the state of the art; each phase of the proposed algorithm is described in increased detail, as well as the user stories that guided the proposed approach, and other practical aspects of the implementation architecture. The prototype was validated with another set of geographical areas of the city of Porto.

The remaining of this paper is structured as follows: Section 2 defines the fundamental concepts related to spider maps, and describes the state-of-the-art methods for their generation. Section 3 details the proposed algorithm. Section 4 describes the architecture of the prototype, which is evaluated and discussed in Section 5. Section 6 concludes this paper.

2. Related work

Transport maps support complex public transport networks by providing essential information, e.g. routes, stops and points of interest [4]. An important process associated with these maps is *schematisation*, where certain aspects are emphasised and unimportant information is removed.

There are several methods for guiding this process. For instance, line generalisation methods, such as

simplification, remove some line points, keeping only those that ensure the overall line shape; *exaggeration* amplifies certain portions of objects; *enhancement* elevate the message and importance of certain features [1]. Another technique adapts the initial map (where points correspond to geographical locations) to a grid [5]. In this technique, line points are moved to grid intersections, while ensuring certain constraints, such as orientation and distance between points. The result is a map with a simpler overall shape, where incremental optimisation processes can be applied to improve the result.

Nonetheless, adapting maps to a grid can lead to very saturated areas, for instance, representing complex centre areas that have lines ending on city outskirts. Sarkar and Brown [6] proposed a method denominated fish-eye that applies different scales throughout the map, thus enabling magnification of crowded areas [7]. This is a *Focus+Context* visualization technique that aids the schematization process, as it emphasises important information while keeping the global context [8].

Spider maps are an effective means of providing information about public transport networks, although few studies addressed this type of maps, in particular how to automate their generation. The majority of the developments in this area relate to the work of João Mourinho [8] in the development of techniques to automate the generation of spider maps.

Spider maps are based on spider diagrams [9] and combine elements from both geographical and schematic maps. These maps are often used to represent complex public transport network, for instance, bus networks in a city centre, and provide passengers information in the pre-planning stage of trips, answering the question "From where I am, where can I go?" [10].

These maps are characterised by a central area, a hub which represents the geographical context [11]. The schematic lines that represent the network routes emerge from the hub. Along with the map, a route finder table is also provided to indicate the direction and route that are associated to each stop within the hub.

The hub is generally depicted by a rectangular shape and details a geographic map of the location, proving the spider map a better spatial context. Around there are located the points that connect the route lines with the stop inside the hub. This corresponds to the points where lines emerge from, hence, their location in the frame should consider route orientation and the stop location within the hub.

Similar to schematic maps, the schematic lines in the spider map do not follow the geographic layout, since they are the result of several simplification and displacement operations [8]. Moreover, spider maps adopt the concept of map point, which describes a relevant point in the map, for instance, stops along the line route, located at a certain canvas coordinates that do not relate to the real geographical location.

Spider maps' schematic lines are defined by a set of segments and map points, some of them shared with different lines, and a start and ending map point. Shared segments are drawn parallel and lines only follow 0, 45 or 90 degrees orientation angles. Segment nodes relate to route stops, however, some stops may be grouped together if they are geographically closed. Moreover, to increase spatial awareness, geographical accidents, such as rivers or seashore, can be added to the map [8].

Spider maps have several other design constraints that should be considered in the generation process. For instance, lines have a certain colour, usually defined by the transport provider, and position of stops and line labels. Similar to schematic maps, spider maps generation is mostly a manual process. However, several techniques for simplification and generalisation can be borrowed from the schematic process. Hence, João Mourinho [8] depicts a set of eight guidelines that spider maps should follow:

1. **Simplification of lines:** Generalise the line as most as possible, while maintaining overall shape.
2. **Group map points:** If several map points are very close together and have similar names, most likely they are related; hence they can be grouped together. This step must be taken carefully, as it can eliminate relevant map information.
3. **Zoom in crowded areas:** Emphasise crowded areas by zooming, which increases readability.
4. **Remove or simplify environment features:** For instance simplify the shapes of geographical accidents.
5. **Group segments that have the same start and end nodes:** draw lines parallel if the segments share the same start and end nodes.
6. **Simplicity over completeness:** depict only the essential elements on the map
7. **Symbol shapes and colours:** use different symbols to emphasise or deemphasise certain map characteristics.
8. **Emphasise important aspects:** direct the user's attention to the most important map aspects, for instance, emphasise the hub to enable a better spatial context to the users.

2.1. Automatic generation of Spider Maps

To the best of our knowledge, very few studies tackled the challenge of automating the generation of

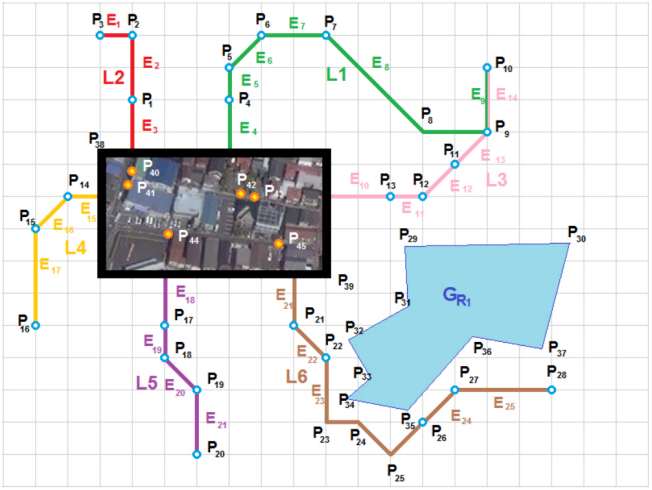


Figure 2. Example of spider map model presented in [8]

spider maps. João Mourinho [8] proposed a method to automatically generate a spider map for a hub location, given a transport network with the same characteristics described above. Since the current general generation process relies mostly on the design expertise and evaluation, the goal is to automatically generate a spider map that preserves topology and ensures the aforementioned restrictions (e.g., line angles).

An important solution this method provides is a complete model representation for the spider map. The spider map *SM* is defined as *SM = (P, V, H, E, L, A, Gr)*, where *P* is a set of map points, *V* the vertices, *H* the hub, *E* a set of direct edges, *L* a set of lines, *A* a set of angles and *Gr* a set of geographical restrictions. Figure 2 depicts a spider map represented by this model.

The initial algorithm state is a geographical accurate map, i.e., map points correspond to the accurate (or similar) geographic location, then a multi-criteria algorithm is applied where the decision variables are the spatial coordinates of each vertex and point belonging to the spider map. The goal is to minimise the objective function while ensuring a set of constraints and design guidelines. Furthermore, two types of constraints are defined: *soft constraints* mostly related to visual qualities and should be followed if possible, and *hard constraints* that ensure a feasible solution and should be enforced. The objective function translates how soft constraints are followed, i.e., if all soft constraints are completely respected, then the objective function is zero, which means the solution is *"optimal"*.

The solution successfully attained the proposed goals. However, this is a complex multi-criteria optimisation problem with great computational effort. For instance, for the default parameters results were obtained in execution times around 5 seconds for simpler maps and 12 seconds for more complex solutions. However, when testing the adjustment of parameters to increase quality, such as the search radius for possible map point displacements, the execution times obtained increased significantly. The quality of the obtained results increased and execution times averaged 14731 seconds [8]. Thus, for a dynamic mobile environment, which has less computational power and should produce results in a shorter time span, this solution needs some adaption, for instance, discarding some constraints.

Ribeiro et al. [12] also proposed a solution to automatically generate a schematic map. Even though it does not fully integrate all the constraints needed for a considerable feasible spider map solution, it proposes a fast solution for the schematic portion of spider maps, which can be an interesting technique for dynamic mobile problems.

The proposed approach is based on the application of force-direct algorithms. Force-direct algorithms are a class of graph algorithms that aim at drawing graphs in an aesthetically pleasing way. Also, this algorithm is computationally lightweight and relatively quick to implement, providing acceptable results, though not optimal. The algorithm's goal is to position nodes so that all edges are approximately equal length, there are as few crossings as possible and objects are distributed uniformly. The method is divided in three steps: initial dataset manipulation, force-directed iterative loop and final adjustments.

Force-directed algorithms apply forces to move nodes to better positions. The algorithm ends when the forces have reached an equilibrium. This approach uses Coulomb's and Hooke's laws to represent, repulsion and attraction forces between nodes, respectively. Additionally, the algorithm implements a set of assumptions and rules, such as two lines intersect if at least a pair of edges intersect, or the flow of execution is dependent on the parametrization.

An important step in this approach is the parametrization, since it influences the quality of the final result. Several parameters are defined, concerning a threshold distance in which nearby nodes are merged, definition of *k* value constant for Hooke's and Coulomb's law and design restrictions, such as pinning start nodes that can be altered to improve the result map.

To test and evaluate the solution, a prototype and an evaluation function were created. Several tests were done with different parameters and total number of nodes. In general, with an average of 15.08 seconds, 95 nodes, 5 lines and 129 edges. The evaluation function combines criteria, such as displacement from original position, line length variation and line crossing variation.

This approach presents relatively fast results and is able to simplify lines, merge close points and present

an aesthetic schematic representation, while preserving the map's topology. However, this solution does not consider many constraints of those spider maps: lines only follow 0, 45 or 90 degree orientations, merge stations (nodes) typically have close names and share the same geographic space and routes that share the same segment are parallel. Furthermore, there is no explanation on how the initial graph is obtained and hub integration is also not considered. Finally, results are linked with parametrisations that are manually provided and change depending on the initial graph.

Finally, in [11] an interactive application was developed that enable the user to select the area of the hub and then the map was produced automatically. Even tough the resultant map could not be designated as a strict spider map representation, it provides a starting point for adapting automatic generation of spider maps with interaction techniques.

2.2. Summary

Transport maps provide passengers an easy way of understanding the underlying network and wayfinding in cities. Spider maps are a type of transport map that combine elements from both geographic and schematics maps. They provide all the travel possibilities from an area (hub) and are typically used in busy city centres where the networks are usually denser.

Such maps are not widely used in comparison with other existing map types. This may be due to the fact that spider maps are mostly manually generated and still rely on the expertise of the designer and stakeholders. Notwithstanding, there are several techniques applied to line generalisation in schematic maps that can be adjusted to assist the spider map generation process.

Moreover, there is not much literature focused on spider maps, and the majority of the efforts made for automating the spider map generation process were done by João Mourinho [8]. Mourinho's solution is able to successfully produce spider maps, however, the goals focus on the quality over performance, making this a complex solution with great computational effort.

3. Automatic generation of spider maps

3.1. Problem definition

The spider map generation process is a complex problem, since these maps have several design constraints as depicted in Section 2. Additionally, the process is mostly done manually, relying on the expertise of the map maker. Even though some current solutions can automatically generate spider maps, they are complex and time expensive for producing results.

Thus, we aim to develop an algorithm capable of producing a spider map by creating, adapting and modifying existing techniques. The solution must take as input the spider map hub area selected by the user and generate as result a viable spider map. A result is considered viable if it satisfies the design restrictions of spider maps aforementioned in Section 2. The goal is to develop a prototype that integrates the developed algorithm capable of producing spider map results in short execution times, since it will affect the prototype usability.

Along with automating the spider map generation process, the prototype should also integrate interaction and visualisation techniques, taking advantage of the benefits of digital maps over the traditional form and thus potentially achieve better usability. Such techniques can be integrated before generating the map, for instance, during the hub selection process, and when visualising the map result, e.g. different levels of zoom and clickable items for additional information.

The developed prototype is focused on Porto city and all the public transport data was provided by OPT[1]. The user is presented a geographic map of Porto for choosing the hub area that will be used as input for generating the spider map. Section 3.2 describes the algorithm for generating a spider map solution.

3.2. Map generation algorithm

The algorithm comprises a sequence of steps that apply displacement operations, to ensure conformance to the spider map restrictions. The pseudo-code of the several steps can be found in Appendix A. The major restrictions are octilinear angles and maintain the topological relations, hence the biggest challenge of the algorithm is to find a location for every map point that ensures octi-linearity, while maintaining the topological relations between points.

Beforehand, the algorithm needs as input the coordinates of the hub, defined by the top left and bottom right corners. These geographical coordinates, i.e., a set of latitude and longitude, will allow querying the server for all the data needed for the spider map generation process. Thus, the server will provide all information related to stops inside the hub and the lines that will belong to the spider map. The lines are defined by a sequence of map points, already established by the database and these will be the points taken into consideration in the algorithm. The goal of the algorithm is to find a location for all map points so that lines follow the spider map design constraints.

Nonetheless, the map points returned from the server are defined by geographic coordinates of their accurate location. Hence, map points need to be projected onto the map canvas, being defined by an x and y instead of

[1] http://www.opt.pt/

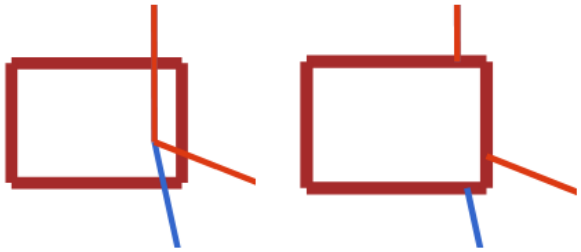


Figure 3. Determination of where lines should emerge from the hub. After hub insertion, the map state is depicted as in the left. The hub on the right demonstrates the calculation of the emerging points and the elimination of segments inside the hub.

latitude and longitude. The map canvas is defined as an SVG using the *D3.js* tool and map points coordinates are projected using the Mercator projection centred at the hub centre. The projection process uses the *D3.js Geo* plugin that provides map projection features.

At this stage, map points have a similar location to their geographic representation, so lines also follow an approximation to the real geographic form. The next step is to insert the hub, defined by the four corners coordinates that represent the boundary area. These coordinates have also been projected so they are defined by an x and y in the canvas. After the hub insertion, the following step is to determine where lines should emerge from the hub and eliminate line segments inside the hub. Therefore, the emerging points, i.e., the points where lines emerge from the hub, will be determined by the intersection of the line segment with the hub. This intersection point represents an approximation of the orientation and path of the line, since the hub is a geographical representation of the area and map points are still located at their original positions, i.e., a close representation of the geographic location. Additionally, all segments positioned inside the hub are eliminated. Figure 3 illustrates this process: the left image exemplifies the map state after hub insertion and the right image demonstrates the calculation of the emerging points and elimination of segments inside the hub.

Map points can be shared by multiple lines and lines can even share segments, thus duplicated information may exist. Hence, the spider map is modelled as a graph $G(V,E)$, where V represents the vertexes, i.e. the map points, and E the edges, i.e. route segments that represent the connection between two map points. Each vertex and edge may belong to one or multiple lines, thus avoiding having duplicated map points or segments. Vertexes have x and y coordinates, a list of lines they belong to, a name representative of the stop or area and an attribute that records if the vertex is an emerging point. It is important to register which vertexes are hub emerging points, since they should not be moved during the algorithm process. On the other hand, edges have two vertexes associated and a list of lines. The order of the map points in the lines is also recorded as well as the colour that lines should be drawn.

Spider maps have associated distortion, since their points do not represent geographical locations, but the product of multiple operations. Furthermore, dimensions are altered, i.e., the hub area is usually augmented and distances between map points are reduced, resulting on a compression effect centred on the hub. However, at this point in the algorithm process, the map dimensions still resemble the real dimensions: the hub is small, and lines are very spread out. Hence, the next step is to resize the hub, increasing its dimensions and translating the lines accordingly. The result of this operation causes the distortion and compression effect aforementioned.

The resized hub dimension was set to 300 by 300 hundred pixels, however, hub selection may not follow this aspect ratio. Hence, the final size of the hub is recalculated so the original aspect ratio is preserved. For instance, if the original hub width is 200 pixels and the original height 100 pixels, the resized hub will have 600 pixels of width and 150 pixels of height.

Nevertheless, after resizing and translating operations, some of the lines may end up intersecting the hub. Thus, map points inside the hub are identified and the maximum distance to the hub boundary is calculated. Then a translation operation corresponding to this distance value is applied, pushing the line out of the hub. At this stage, the map is similar to the original, but with distortion and with lines closer to the hub boundaries.

The hub is a portion of a geographic map that depicts the area associated with the spider map. To obtain the geographic map image Here API[2] was used, providing services that return a map image of the specified area. The image already has the correct size, i.e., same size as the hub, thus it is placed on the hub coordinates. Figure 4 illustrates an example of a hub of Casa da Música surroundings. The markers represent the stops in that area.

Furthermore, before beginning the displacement operations to satisfy the spider map restrictions, a matrix containing the topological relations between points is built. It is important to build the matrix before the generation process starts, since at this stage all the points relate to each other close to their real geographical location. Thus, for each map point is calculated the relation to every other map point. A map point can be north of (No) or south of (So) and east of

[2]https://developer.here.com/

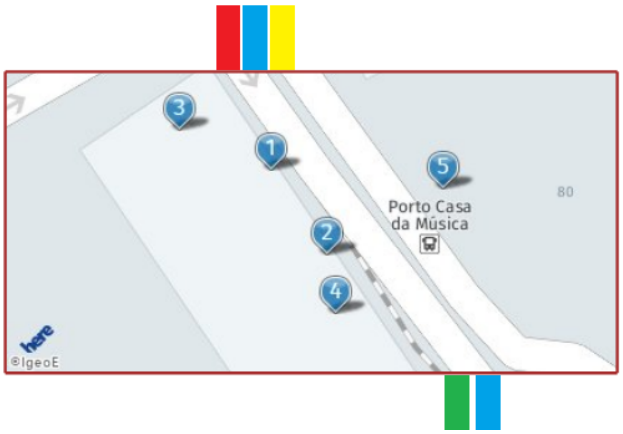


Figure 4. Hub example of Casa da Música area

Table 1. Example of topological relation matrix between points P1, P2 and P3

-	**P1**	**P2**	**P3**
P1	-	inLine, Wo	No, Wo
P2	inLine, Eo	-	No, inLine
P3	So, Eo	So, inLine	-

(Eo) or west of (Wo) another point. When two points have an equal coordinate (x or y), they are defined as *in line* of each other. Table 1 depicts the topological matrix of points P1, P2 and P3 relations.

After this step is completed, the displacement operations begin in order to find a location for every map point that satisfies spider maps restrictions. First, a grid adaptation operation is done with the intent of simplifying the overall shape of lines. Next, all angles are ensured to be octilinear and then the spider map is displayed, following the draw rules. The next sections depict these algorithm steps, that will displace map points trying to generate a viable map solution.

1) Grid adaptation. The first step of the algorithm is to adapt the current map to a predefined grid, by assigning a grid intersection point to every map point. This step simplifies the overall shape of lines, leading to a closer solution where spider map restrictions are followed.

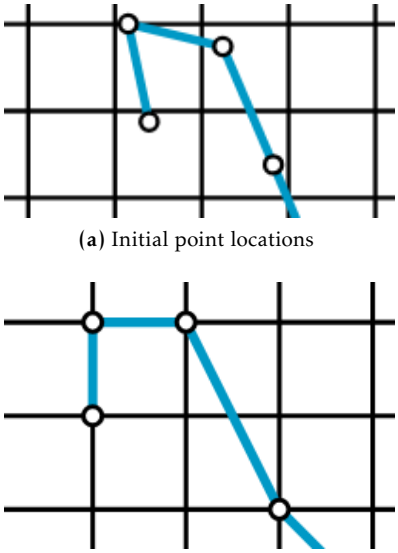
The first task is to build the grid over the map, thus the maximum and minimum x and y values of the map points are determined which represent the bounds of the map and, subsequently, the boundaries of the grid. Then, the grid is built with an initial grid cell size of 20 pixels by 20 pixels. It is important to note that the cell size will affect the complexity of the algorithm, since grid cells with smaller cell size lead to finer grid granularity, which increases the search for possible displacements. On the other hand, it may not be possible to adapt a map to a grid if cells have a large size, given that possible displacements will be scarce. After several tests with different sizes, this initial cell size was chosen since results showed that most maps could successfully adapt to a grid with this cell size, without the need of repeating the process by adapting the cell size. The final step in building the grid is to determine the grid intersection points. These points represent all the possible displacement for map points during the grid adaptation process and the subsequent algorithm steps.

Therefore, for every map point the nearest grid intersection points are calculated, i.e., the 16 surrounding and closest grid intersection points are determine. However, not every nearest grid intersection point is a valid displacement. A grid intersection point is considered for a valid displacement if it causes no hub occlusions, i.e., does not cause any segments to intersect with the hub; does not lead to any segment overlapping, i.e., segments do not pass through map points that do not belong to that segment; and the grid intersection point is free, i.e., it does not have a map point assigned.

The grid intersection point selected for the displacement is the one with the smallest score, which represents the attribution of less penalties. The score combines the distance from the map point to the grid intersection point being evaluated (points with greater distance will be more penalised) and a score that translates how well topology relations are maintained, by giving a penalty to every topological violation. A displacement causes a topological violation if it changes the relation between two points. For instance, having P1 south of and east of P2, a displacement that leads to P1 being north of or west of P2 is considered to cause a topological violation. Smaller penalties are given to displacements that cause relations to change to in line. This trade off by loosening the topological constraints leads to simpler line shapes, where lines become straighter which causes less non-octilinear angles. If no topological violation occurs, then the topological score given is zero.

However, in some cases it may not be possible to adapt the map to the grid with the current grid cell size. This means that some map points may not have any possible valid displacements. Hence, the grid cell size is decreased, the map points coordinates are restored to their original locations and the grid adaptation process is restarted. Decreasing the cell size leads to a finer grid granularity, which in turn leads to more possible displacements. This processed is repeated until the grid adaptation is successfully completed or the grid cell size reached a defined minimum. In this last case, the grid adaptation process may not be possible, thus a map solution will not be produced.

Figure 5 depicts the grid adaptation process, illustrating initial locations in the top image and the displacement result in the bottom image. In the figure



(a) Initial point locations

(b) Displaced point locations

Figure 5. Grid adaptation process. The initial locations in (a) are displaced to new locations on the grid as in (b), yielding simpler shapes. Some of the angles become octilinear as a result of this process.

is possible to note that lines have a simpler shape and some of the angles already comply with the octilinear angle restriction.

However, not all the nearest grid intersections are valid displacements. Grid intersection points that will cause hub occlusion, i.e., will intersect the hub, and that will cause line segments to overlap or pass through map points that do not belong to that segment are removed as possible displacement locations. The addition of this restriction will lead to, in some cases, map points that will not have any possible displacements. When this happens, the graph is returned to the original state, the grid cell size is decrease and the grid adaptation process is restarted. By decreasing the grid cell, the granularity is increased which leads to more displacement options. This process is repeated until all points are displaced to a grid intersection or the grid cell size reaches a defined minimum. In this last case, the grid adaptation process may not be possible, thus a map solution will not be produced.

2) Correcting non-octilinear angles. After the grid adaptation process is finished, the result is a map with simpler line shapes and where map points respect the topology relations. However, some of the lines may still not follow octilinear angles. Just as mentioned in Section 2, one of the spider maps restrictions is that angles should only be of 0, 45 or 90 degrees, i.e., only octilinear angles. Hence, the next algorithm step is to identify and correct non-octilinear angles.

The first step is to identify all the map points where two segments form a non-octilinear angle. Map points corresponding to hub emerging points are not taken into consideration, since they will be approached using a different method to ensure the lines also form octilinear angles when intersecting the hub boundaries.

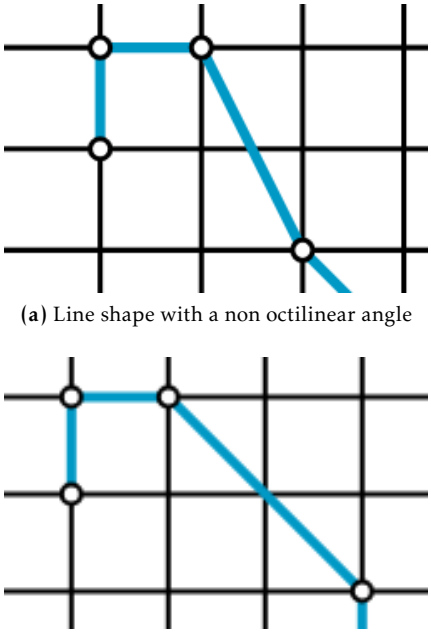
Afterwards, for each map point identified with an incorrect angle, the algorithm will try to identify a grid intersection point which displacement will correct the angle. The process is similar to the nearest grid intersection points search in grid adaptation, where the closest grid points are identified and invalid displacements are removed from possible options. A grid intersection is considered not valid for non-octilinear angle correction if:

1. The displacement will cause octilinear angle to become non-octilinear;
2. The displacement will disturb topological relations between points (changes to in line are not considered as disturbance);
3. The displacement will cause occlusions with the hub, line overlapping or lead to segments passing through map points that do not belong to that segments;
4. That grid point already was a map point associated;
5. The displacement will cause the angle to remain non-octilinear.

Just as in grid adaptation, scores are calculated for every valid option and the map point is displaced to the best scored grid intersection point, i.e., the grid intersection point with the smallest score. Figure 6 exemplifies the correction of a non octilinear angle by displacing a map point to another grid intersection point.

Nonetheless, some map points will not have any possible valid displacements that will correct the non-octilinear angles, thus making them candidates for a break point introduction. A break point is a map point introduced in one of the segments of the incorrect angle to correct the non-octilinear angle without displacing any map point. This new map point is added to the graph representation and marked as being a break point. Break points are introduced to correct angles and have no other meaning to the spider map, so they need to be represented differently.

To introduce a break point, grid intersections surrounding the identified segments are searched and checked if the displacement will correct the angle. Similar to the previous operations, a displacement is valid if causes no occlusions, and no segments overlap. A break point introduction will transform one segment



(a) Line shape with a non octilinear angle

(b) Line shape after point displacement

Figure 6. Line shape before (a) and after (b) the correction of a non-octilinear angle by map point displacement

in two new ones, allowing angles to be corrected. Figure 7 illustrates a result of a break point introduction.

Figure 7. Non-octilinear angle correction with segment break point

Although introducing a break point will correct most of the remaining non-octilinear angles, in some cases, mostly in very dense areas, it is not possible to find a valid location to introduce the break point. Hence, two break points are introduced to correct these last cases. The introduction of two break points is very similar to inserting a single one. However, the goal is to find the best combination of two valid grid points that can correct the angle. The complexity of this problem is restrained by limiting the grid points search to the nearest grid points and by eliminating all the invalid grid locations. Figure 8 depicts non-octilinear angle correction by inserting two break points.

Moreover, lines emerging from the hub should also make an octilinear angle with the hub boundaries and, just as aforementioned, the correction of these angles

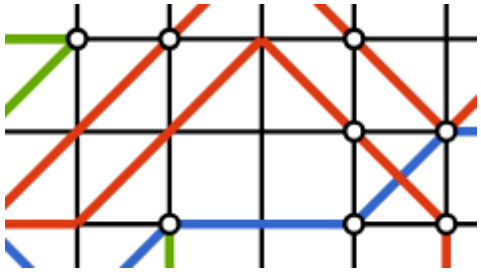


Figure 8. Non-octilinear angle correction with two break points

is treated separately from the remaining map points. To correct the angles from hub emerging segments is established that those segments should make a 90º angle with the hub boundary. Then, a break point is introduced in that segment, so the corresponding angle is 90º or, if not possible, 45º degrees. Figure 9 depicts the correction of angles from hub emerging segments.

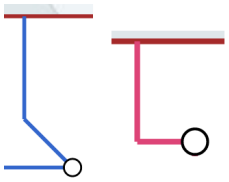


Figure 9. Correction of non-octilinear angles of hub emerging segments

3) Draw the spider map. After correcting the angles, the final map point locations are determined, and the drawing process can begin. The first is to obtain and place the hub image, that is a geographical representation of the area. The image is obtained using the API Here[3] that returns an image of the geographical map giving a boundary box. Moreover, the stops are identified with markers.

Map points are drawn in the associated locations and do not need further processing. However, segments are shared between lines and need to be drawn parallel, thus making it necessary introducing an offset between shared segments.

In order to introduce an offset that will lead to parallel segments, it is necessary to calculate the slope of the line. Thus, identifying the correct orientation (vertical, horizontal or diagonal), is possible to introduce a correct offset to the x and y coordinates, just as illustrated in Figure 10.

The final step is to draw the labels that identify the map points. Not all map points need to be labelled, only the last and the most important of

[3]https://developer.here.com/

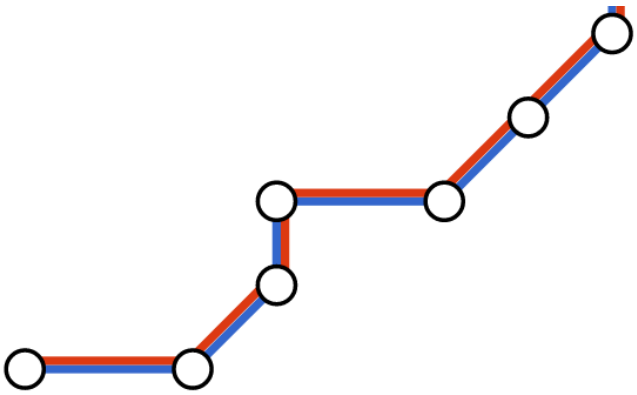


Figure 10. Shared line segments

each line. Nonetheless, the label's position needs to be determined. Thus, a score is calculated that translates how many occlusions will the label cause. For that, a bounding box of the label is placed at top, bottom, left and right of the corresponding map point and a penalty score is given for each line intersection. The chosen place will be the one with the smaller score.

After this step, the generation process is finished and a valid spider map solution is presented to the user. Figure 11 depicts the algorithm generation process in a flowchart. Hence, a valid spider map solution is generated if all the aforementioned algorithm steps are successfully completed. In some cases the algorithm is not capable of producing a valid solution, for instance, if grid adaptation fails, no solution will be presented, or if not every non-octilinear angle is corrected, the spider maps will have errors.

The developed algorithm is integrated in the developed prototype depicted in Section 4 and results will be illustrated and evaluated in Section 5.

4. Prototype development

For the purpose of testing how the developed algorithm performs in real situations, a prototype was created integrating the algorithm depicted in Section 3.2 and taking advantage of digital map characteristics by combining visualisation and interaction techniques.

4.1. Use cases and stories

The main use cases consist of selecting the desired hub area, and generating the respective spider map. A set of user stories was defined (see Table 2); they cover the main functionalities that such a prototype should implement. Moreover, there is the ambition to integrate interaction and visualisation techniques to enhance the user experience.

In the first screen, a map of Porto city with interaction capabilities is presented to the user, i.e., the user can

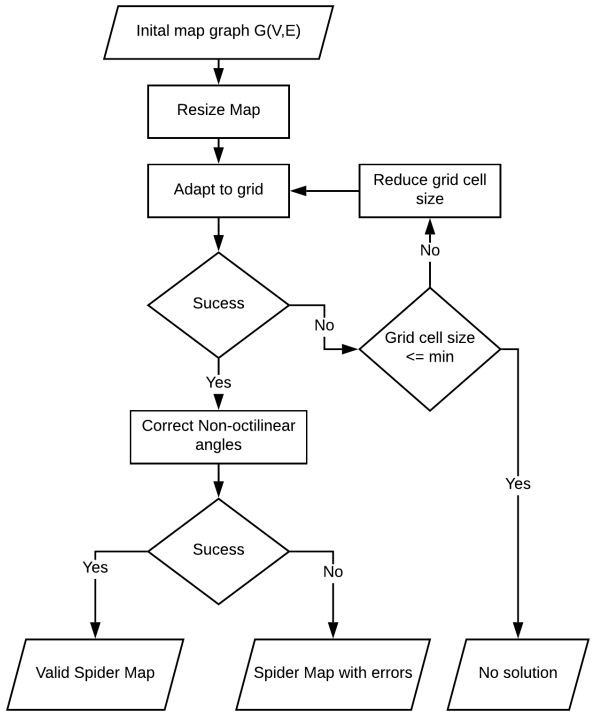


Figure 11. Algorithm workflow for the generation of a spider map. The corresponding pseudo-code can be found in Appendix A.

Table 2. User stories defined for the functional prototype

User story	Description (As a User, I would like to...)
US01	See and navigate a map of Porto city area
US02	See a pre-defined grid that marks possible hub placements
US03	Choose one or multiple grid cells that define the hub area
US04	Clear current grid selections
US05	Check the stops inside the hub selection
US06	Check additional information about stops
US07	Generate a spider map given the hub selected in the grid
US08	Visualise the spider map resulted from my hub selection
US09	Check information about the hub stops and lines that belong to the spider map
US10	Interact with the spider map by zooming, moving and clicking on elements for additional information

zoom and navigate through the map. Furthermore, in the top right corner, the user can access control buttons illustrated in Figure 12 left. In this controls users can

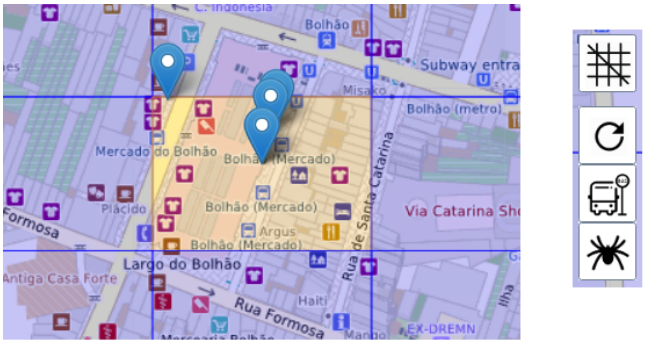


Figure 12. Example of grid selection (left) and control buttons (right)

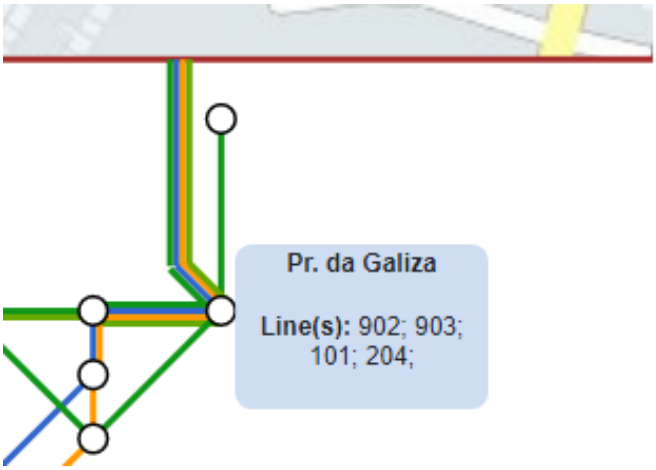


Figure 13. Interaction mechanism within the spider map: clicking on map points reveal additional information

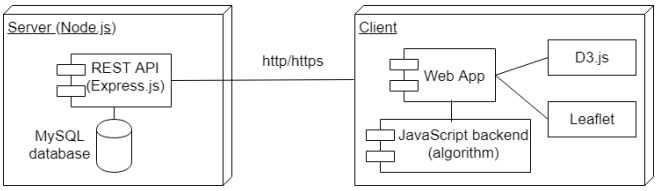


Figure 14. Solution Architecture for the prototype

show/hide the pre-defined grid, check stops inside the grid selection and finally generate the spider map.

The pre-defined grid lays over the Porto city corresponding to the boundaries of the available data. Then, the user can select one or combine several grid cells to create a personalised hub area. As the grid has an arbitrary size, it could have been possible to previously generate a spider map for each grid cell in advance, to reduce the user's waiting time for navigating a map. However, such assumption would imply that the user could not select more than one grid cell, thus limiting the user's capability of defining a region of interest that may span a number of adjacent grid cells. Figure 12 right shows an example of grid selection, where selected cells are shown in orange and markers depict stops inside the hub selection.

After the user chooses the desired hub and selects "*Generate Spider Map*", the algorithm takes the hub coordinates as input and generates a spider map result. In the next screen the user can visualise and interact with the map result. The user can navigate, zoom and click on map points to check additional information. All these interaction features were developed using *D3.js Behaviour* plugin that allow to catch and handle interaction events. Figure 13 depicts an example of a portion of a spider map result where it is possible to check the additional information box when a map point is hover or clicked on.

4.2. Architecture

The developed solution follows a simple two-tier architecture or client-server, illustrated in Figure 14. This architecture style is commonly used in distributed systems to separate operations into the client and server, where the server provides services to the client [13]. Thus, the server is responsible for dealing with all the necessary data operations, while the client is responsible for the spider map generation and rendering operations.

The server was implemented using *Node.js*[4] and *Express.js*[5] for the REST API. Moreover, the server establishes a connection with a *MySQL*[6] database that stores all the public transport data, that will be depicted in detail in the next section. Hence, the server is responsible for gathering and processing all the data needed for the spider map generation. The architecture was built and tested on a mid-range 2019 laptop.

The implemented API has two main models – routes and stops – with several endpoints for handling and retrieving information associated with each one of the models, as described in Table 3.

On the other hand, the client consists of a web application based on the two major use cases described in Section 4.1. For the geographic maps and hub selection *Leaftlet*[7] and *OpenStreetMap*[8] were used, while the spider map drawing and generation was developed using *D3.js*[9] and *Javascript* technologies.

[4]https://nodejs.org
[5]https://expressjs.com/
[6]https://www.mysql.com/
[7]https://leafletjs.com/
[8]https://www.openstreetmap.org/
[9]https://d3js.org/

Table 3. API Endpoints for retrieving input data for the generation of a spider map

Endpoints	Parameters	Description
/stops	-	Get all stops
/stops/hub	topLong; topLat; bottomLong; bottomLat	Get stops within a rectangular hub defined by two pairs of geographical coordinates
/stop	stopID	Get stop by ID
/stop/lines	stopID	Get routes of all of the lines that go through a stop
/line/stop	stopID	Get all lines that serve a stop
/line	lineID	Get line by ID
/line/code	code	Get line by code
/line/route	lineID	Get line route by line ID

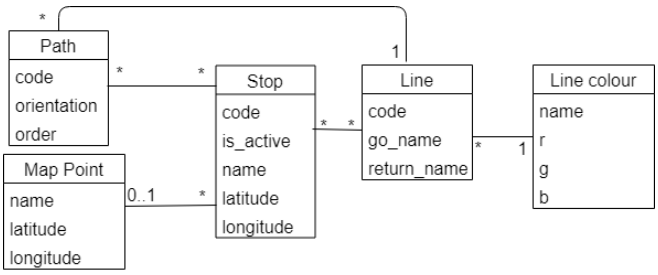


Figure 15. Data model for the structural elements of a spider map

4.3. Data model

The data related to public transport network of Porto was provided by OPT and stored in a MySQL database following the model depicted in Figure 15.

Lines are characterised by a code, name and a line colour. Stops are defined by a code, name and geographic coordinates (latitude and longitude). Lines consist of several stops; each stop may belong to zero or more lines. Nonetheless, routes are defined by the table "Path" that define the sequence of stops identified by the attribute "order". The data provided already defines map points that may represent groups of stops. Henceforward, when a stop is associated with a map point, it should be replaced when forming the path of a line.

Furthermore, stops and map points have associated geographical coordinates (latitude and longitude), which will represent the initial position of the points. Moreover, lines and stops also have other attributes associated, such as names and line colours.

5. Evaluation and validation

Current solutions are complex and take very long to produce spider map results. Hence, the ambition is to tackle the complexity of the generation process of spider maps and develop a solution capable of automatically generate spider maps in real-time. Thereby, the two variables taken into consideration during the evaluation and validation are if the map is correctly generated, i.e., the spider map follows the establish design rules, and the execution time needed to produce the result. A result is considered valid if it complies with the spider map restrictions aforementioned in Section 2.

5.1. Tests and results

Performed tests aim at testing if the solution is capable of generated valid spider maps at real-time using the prototype develop to select the input hub area and generate and evaluate map results. Several tests were performed by choosing different hub areas as input and evaluating the results. Even though tests were only performed for Porto's bus network, the number of possible hub inputs is extensive. Hence, the tests focused on testing areas where the network is denser, i.e., areas served by many public transports' lines like city centres. In Porto, some of the busiest areas are *Aliados*,*Casa da Música*, *Hospital São João*, *Castelo do Queijo*. Tests demonstrated that the developed algorithm produces feasible spider map results for the city of Porto. Figures 16 depicts the initial map state for *Castelo do Queijo*, a coastal area, and Figure 17 depicts the corresponding spider map. Figure 18 also depicts a spider map result for *Aliados*, a busy centre area in Porto.

The complexity of the generation process and, subsequently, the spider map is directly related to the number of map points, i.e., the complexity increases as the number of map points also increases, since more displacement operations and angle corrections will be needed to generate a valid map. Hence, to control the continuous increase in complexity, a limit to the number of lines in the spider map was set, as well as a limitation on the hub size. This prevents the user to select large areas for the algorithm, preventing the exponential increase in complexity.

There is not much literature in automating the generation process of spider maps, and most of the efforts made in this area were through Mourinho's [8] work. However, in his work the goal was to find the optimal spider map solution, hence the quality was valued over fast results. Thus, the solution required great computational effort and long execution times. For instance, in tests accessing the quality versus the number of algorithm iterations, the average execution times were of 2797 seconds.

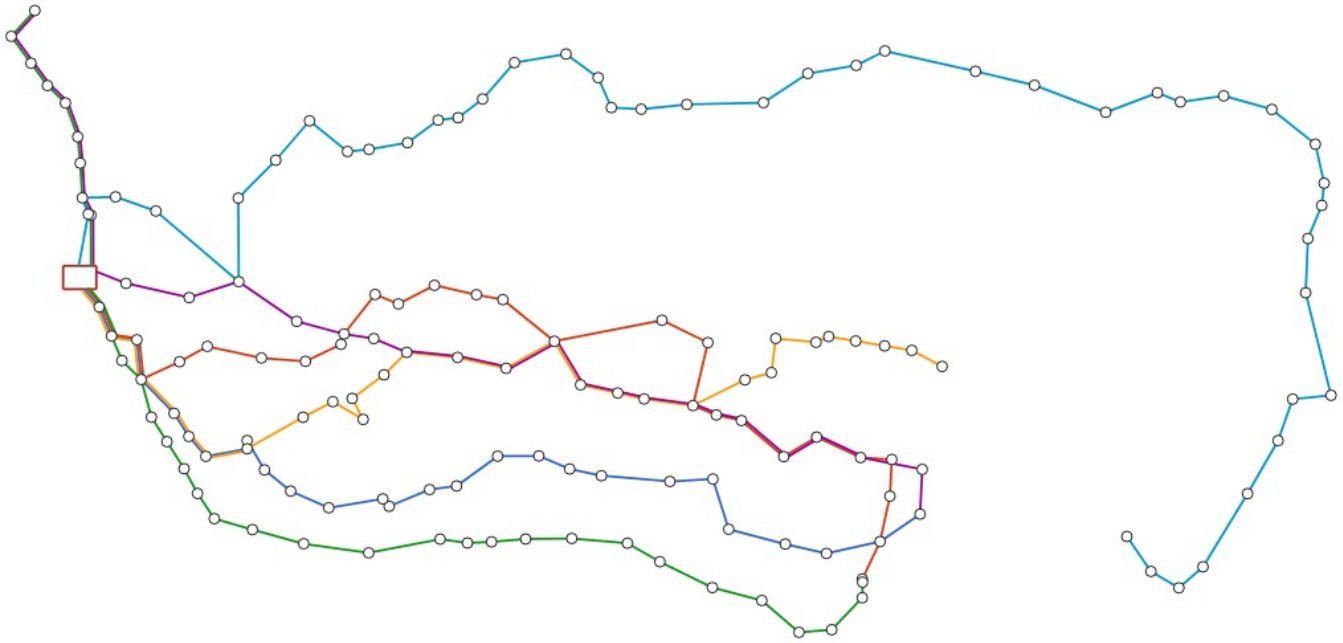


Figure 16. Initial map state for *Castelo do Queijo* hub area

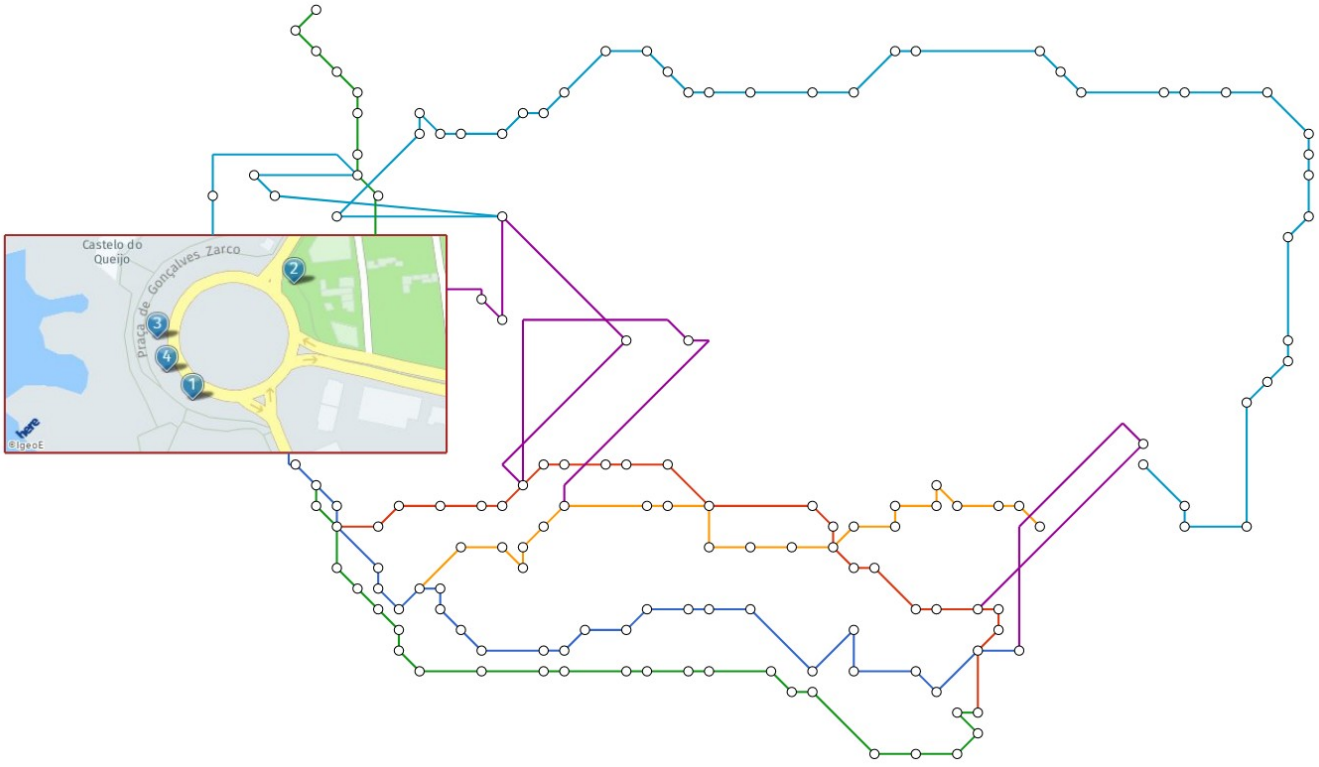


Figure 17. Spider Map result for *Castelo do Queijo* hub area

Even though is not possible to establish a direct comparison with the tests performed by Mourinho, it is possible to conclude that the developed solution was able to produce results faster. The developed solution produced spider maps under 500 milliseconds for complex centre areas. Table 4 depicts tests results for

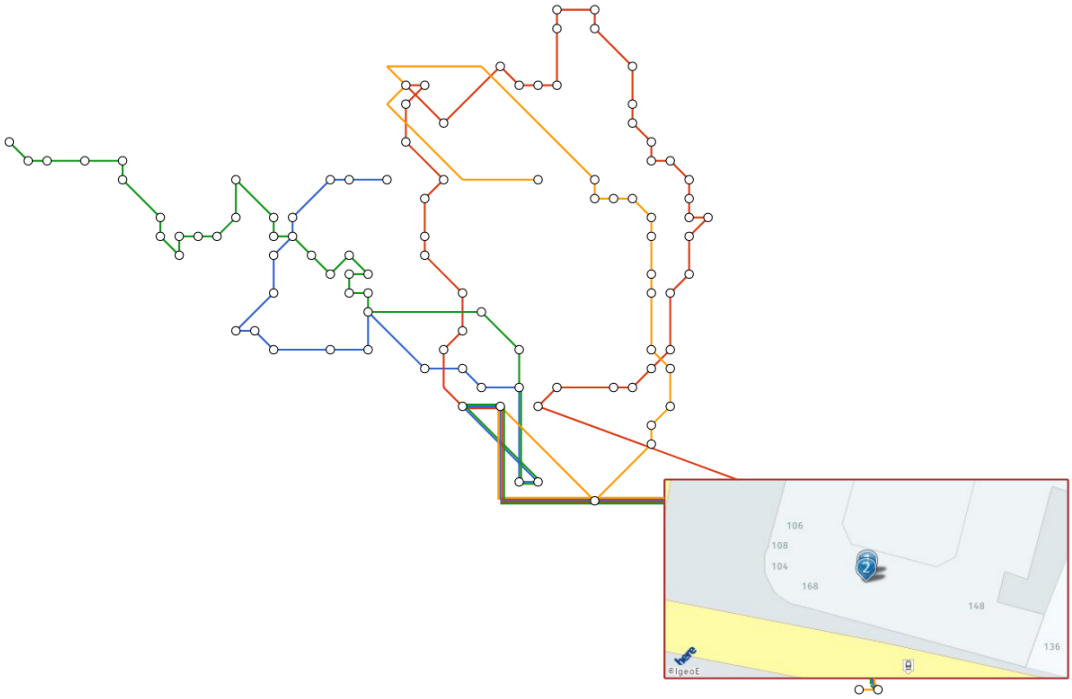


Figure 18. Spider Map result for *Aliados* hub area

valid solutions, describing the number of map points, the numbers of different lines of the map, hub area and the execution time (ET) in milliseconds. In addition, Table 5 depicts the success of test results, identifying how frequently a valid solution was obtained, the number of times where a solution was not possible and the number of incorrect solutions (i.e., spider map results that contain some non-octilinear angles).

It can be concluded that the developed algorithm successfully produces results, i.e., the solution generates valid spider map results in real time, taking significantly less time compared to state-of-the-art solutions. Thus, this work successfully tackles the complexity of the spider map generation process and contributes to the identified gap of current work.

5.2. Limitations and Future Work

The quality of the result depends on how and if all the stages of the algorithm are successful. In the grid adaptation stage, the algorithm will adapt the cell size until the initial map is successfully adjusted to the grid; however, in some cases, grid adaption may not be possible. In dense areas, a vast number of map points compete for a grid allocation. Thus, even by increasing the grid granularity, it may not be possible to assign a grid point to every map point. Moreover, since the subsequent algorithm steps depend on the success of grid adaptation, a solution may not be found. Nevertheless, the introduction of a constraint to the

Table 4. Tests results for generated spider maps

No. map points	No. lines	Hub area	ET (ms)
153	6	Castelo do Queijo	844.29
153	6	Castelo do Queijo	710
32	1	Av. Boavista	122.21
153	6	Casa da Música	419.23
144	1	Casa da Música	298.64
10	1	Aliados	35.04
108	4	Aliados	387.71
130	4	Trindade	664.88
52	2	Boavista	215.83
88	2	Hosp. São João	272.87
106	4	Hosp. São João	432.53
102	4	Av. Boavista	5445
32	1	Av. Boavista	102.67
105	4	Praça da República	482.9
105	4	Aliados	496.27
27	1	Bolhão	95.45
39	1	Campo Lindo	177.45
66	3	Marquês	244.62
177	6	Marquês	599.46
37	6	Passeio Alegre	105.96
51	6	Foz do Douro	159.23
33	6	Ramalde	102.33
79	6	Parque Real	194.93

Table 5. Outcome of performed tests

Solution	No. of results
Valid solution found	22
No solution found	3
Solution with errors	5

maximum number of lines overcame this problem, and tests showed that the grid adaptation process is successfully completed even in complex areas, and with just one or two iterations. Therefore, reducing the cell size in each iteration to increase the grid granularity was proven adequate. Hence, it is likely that users may need to select one or more grid cells in order to define the desired geographic area.

The next algorithm step that will influence the quality of the solution is the correction of non-octilinear angles. In the developed solution, the algorithm has several iterations that aim correcting the non-octilinear angles through several approaches. The first approach is identifying a valid grid allocation to displace the identified map points and correct the angle. Nevertheless, in some cases is not possible to find a valid displacement that corrects the angle, thus the next iterations try to correct the remaining non-octilinear angles by inserting one or two break points. The integration of different approaches to correct identified non-octilinear angles was effective in producing valid spider map results.

Notwithstanding, in some cases the algorithm may not produce a valid spider map (i.e., some angles may not be corrected) or, in the worst-case scenario, not produce a solution. Most invalid algorithm results derive from the non-octilinear angle correction, not the grid adaption as depicted in Table 5. Thus, even for invalid results, the algorithm can present a solution that may not be completely correct (some angles may not be octilinear).

Some errors are the result of incorrect map point coordinates, that lead to incorrect projections, which subsequently cause the failure of grid adaptation or angle correction.

On the other hand, circular lines are viewed as a special case, since they sometimes lead to particular results. For instance, results with circular lines often cross themselves, which may be valid according to spider map restrictions, but is not aesthetically pleasing. Also, scaling the hub may lead to undesired distortion, that in some cases may preclude the success of the non-octilinear angle correction.

Even though some limitations were identified and the algorithm may be improved so it becomes more robust to certain cases, results have proven that the developed solution was successful and provides enhancements in the current state-of-the-art solution. The solution is able to produce viable spider map solutions at real-time and taking in consideration the hub area as user input, whilst maintaining a feasible quality that allows users to explore the transportation network. Furthermore, the prototype demonstrated the successful integration of the algorithm with the advantages of digital maps by incorporating visualisation and interaction techniques.

Finally, the spider map solutions can be aesthetically improved in a post-processing stage, with more line simplifications. Nonetheless, the developed solution provides advances in the simplification of the generation process of spider maps, thus potentially making an impact on the use of spider maps in providing public transports information. Through the developed prototype, the user is able to choose a desire hub area and visualise all the travel possibilities by the generated spider map.

6. Conclusions

Spider maps are a type of transportation map that presents all the public transport possibilities available within a geographic area. These maps allow one to identify how far they can go by using the available transport network service within that area. However, their production is still mostly manual, and tailored to a restricted set of locations. Some efforts have been made to automate the generation of these maps, but state-of-the-art solutions require great computational effort and long execution times to produce results. Hence, the proposed approach aimed at developing a solution capable of automatically generating spider maps, tackling the complexity gap of current solutions.

This work focused on two goals: developing an algorithm that automatically generates spider maps results in real-time and considering a hub that can be defined by the user according to the desired area, regardless of a cell grid size defined by the system's implementation; developing a prototype that integrates the map generation algorithm, adding interactive capabilities to map results. The algorithm adapted techniques used in schematic maps generation, such as adapting a map to a pre-defined grid, and developed new processes that apply several operations to produce a spider map compliant to all the design restrictions.

Throughout the validation and evaluation process, the objective was to test if the solution could produce feasible spider map solutions at real-time, reducing the execution time needed to produce map results whilst correctly depicting the information about the bus network. The prototype and tests focused on the bus network of Porto.

Performed tests showed that the solution is successful and can produce map results in shorter execution times than state-of-the art solutions. Furthermore, the prototype developed validated that the algorithm can

be incorporated into a web or mobile application that provides an interface for passengers to interact and customise the map generation.

Future work may improve the map aesthetics in a post-processing phase by applying more simplification to the spider map schematic lines, which will increase the quality of the solution, and other algorithm improvements so it becomes more robust to complex network data. Notwithstanding, the developed solution contributed to the identified gap in state-of-the-art solution, producing spider map solutions at real-time and considering user input.

Acknowledgements. This work is financed by the ERDF - European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT-Fundação para a Ciência e a Tecnologia within project PTDC/ECI-TRA/32053/2017 and POCI-01-0145-FEDER-032053. We also acknowledge OPT[10] for providing the data related to Porto's bus network.

References

[1] Klippel, A., Richter, K.F., Barkowsky, T. and Freksa, C. (2005) The cognitive reality of schematic maps. *Map-based Mobile Services: Theories, Methods and Implementations* : 55–71doi:10.1007/3-540-26982-7_5.

[2] Sociedade de Transportes Colectivos do Porto (2019), São João hospital spider map, https://www.stcp.pt/fotos/spider_map.

[3] Santos, S., Dias, T.G. and Sobral, T. (2020) Automatic generation of spider maps for providing public transports information. In Martins, A.L., Ferreira, J.C. and Kocian, A. [eds.] *Intelligent Transport Systems. From Research and Development to the Market Uptake* (Cham: Springer International Publishing): 131–149.

[4] International Cartographic Association (2019), History of ICA, https://icaci.org/research-agenda/history/.

[5] Klippel, A. and Kulik, L. (2000) Using grids in maps. *Theory and application of diagrams. First International Conference, Diagram 2000, Edinburgh, Scotland, UK, September 1-3, 2000 Proceedings* : 486–489.

[6] Sarkar, M. and Brown, M.H. (1992) Graphical Fish-eye views of graphs. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* : 83–91doi:10.1145/142750.142763.

[7] Baudisch, P., Good, N. and Stewart, P. (2001) Focus Plus Context Screens - Combining Display Technology with Visualization Techniques. *Proceedings of the International Symposium on User Interface Software and Technology (UIST'01)* : 31–40doi:10.1145/502348.502354, URL http://doi.acm.org/10.1145/502348.502354.

[8] Mourinho, J. (2015) *Automated Generation of Context-Aware Schematic Maps: Design, Modeling and Interaction*. Ph.D. thesis, Faculty of Engineering of University of Porto. URL https://hdl.handle.net/10216/79324.

[9] Avelar, S. and Hurni, L. (2006) On the Design of Schematic Transport Maps. *The International Journal for Geographic Information and Geovisualization* **41**(3): 217–228. doi:10.3138/A477-3202-7876-N514.

[10] Maciel, F. and Dias, T.G. (2016) Challenging user interaction in public transportation spider maps: A cobweb solution for the city of Porto. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* : 181–188doi:10.1109/ITSC.2016.7795551.

[11] Maciel, F.M.A. (2012) *Interactive Spider Maps for Public Transportation*. Ph.D. thesis, Faculty of Engineering of University of Porto.

[12] Ribeiro, J.T., Rijo, R. and Leal, A. (2012) Fast Automatic Schematics for Public Transport Spider Maps. *Procedia Technology* **5**: 659–669. doi:10.1016/j.protcy.2012.09.073, URL http://linkinghub.elsevier.com/retrieve/pii/S221201731200504X.

[13] IBM (2019), The Client/Server model.

Appendix A. Pseudo-code for the spider map generation algorithm

[10] http://www.opt.pt/

Algorithm 1: Resize initial map

```
Input  : Hub corners coordinates; map graph
         G(V,E)
Output: Resized hub coordinates and graph
         G(V,E)
calculate hub width, height and centre from
 coordinates;
newWidth ← 300 * width / height;
newHeight ← 300 * height / width;
scalingFactorX ← newWidth / width;
scalingFactorY ← newHeight / height;
for each v ∈ G(V,E) do
    if v is hub emerging point then
        calculate vector from hub centre to v;
        calculate new v coordinates applying a
         translation of the obtained vector with the
         scaling factor;
    else
        identify the corresponding line hub emerging
         point;
        apply same translation of the identify hub
         emerging point;
    end
end
lines ← GetLines(E);
for each line ∈ lines do
    if line is inside then
        calculate maximum distance to hub
         boundaries;
        apply translation to every line map point;
    end
end
```

Algorithm 2: Adapt map to grid

```
Input  : Map graph G(V,E)
Output: Map graph G(V,E) adapted to grid
for each v ∈ G(V,E) do
    if v not hub emerging point then
        calculate nearest grid intersection points;
        for each nearest grid point do
            if grid point is valid then
                calculate grid point score;
            end
        end
        if valid displacements found then
            update v coordinates to grid point with
             best score;
        else
            if cellSize < min then
                solution not found;
            else
                decrease cell size;
                reset graph to original locations;
                restart GridAdaptation;
            end
        end
    end
end
```

Algorithm 3: Correct non-octilinear angles with displacement

```
Input  : Map graph G(V,E)
Output: Break point candidates
incorrectVertexes ←
 FindNonOctilinearAngles(V);
for each v ∈ incorrectVertexes do
    if v not hub emerging point then
        calculate nearest grid intersection points;
        for each nearest grid point do
            if grid point is valid then
                calculate grid point score;
            end
        end
        if valid displacements found then
            update v coordinates to grid point with
             best score;
        else
            add v to break point candidates
             breakPointCandidates;
        end
    end
end
```

Algorithm 4: Correct non-octilinear angles with break point insertion

```
Input  : Map graph G(V,E), break point
         candidates breakPointCandidates
Output: Two break point candidates
for each v ∈ breakPointCandidates do
    if v not hub emerging point then
        calculate nearest grid intersection points;
        for each nearest grid point do
            if grid point is valid then
                calculate grid point score;
            end
        end
        if valid grid intersection gridPt found then
            add break point gridPt to V;
            transform edge E(pt1,pt2) into
             E1(pt1,gridPt) and E2(gridPt,pt2);
        else
            add v to two break point candidates
             twoBreakPointsCandidates;
        end
    end
end
```

Algorithm 5: Correct non-octilinear angles with two break points insertion

```
Input  : Map graph G(V,E)
Output: Map graph G(V,E)
for each v ∈ twoBreakPointsCandidates do
    if v not hub emerging point then
        calculate valid nearest grid intersection
         points;
        for each nearest grid point do
            calculate two grid point combination;
            calculate score;
        end
        if valid two points grid intersection
         combination gridPt found then
            add break points gridPt1 gridPt2 to V;
            transform edge E(pt1,pt2) into
             E1(pt1,gridPt1), E2(gridPt1,gridPt2)
             and E3(gridPt2,pt2);
        else
            G(V,E) with non-octilinear angles;
        end
    end
end
```

Algorithm 6: Correct non-octilinear angles of hub emerging segments

```
Input  : Map graph G(V,E)
Output: Map graph G(V,E) with correct hub
         emerging segments
for each v ∈ G(V,E) do
    if v is hub emerging point then
        find grid intersection for 90º angle;
        if grid intersection not found then
            find grid intersection for 45º angle;
        end
        insert break point at grid intersection found;
    end
end
```

Algorithm 7: Generate Spider Map

```
Input  : Hub coordinates
Output: SVG of spider map
Retrieve route information from database;
build map graph G(V,E);
adaptToGrid();
if grid adaptation successful then
    correct non-octilinear angles;
else
    reset graph G(V,E) to original coordinates;
end
drawSpiderMap();
```