Supervised Force Directed Algorithm for the Generation of Flow Maps

Alberto Debiasi
Fondazione Graphitech
Via alla Cascata 56/C
38123, Trento, Italy
alberto.debiasi@graphitech.it

Bruno Simões
Fondazione Graphitech
Via alla Cascata 56/C
38123, Trento, Italy
bruno.simoes@graphitech.it

Raffaele De Amicis
Fondazione Graphitech
Via alla Cascata 56/C
38123, Trento, Italy
raffaele.de.amicis@graphitech.it

ABSTRACT

Cartographic flow maps are graphical representations for portraying the movement of objects, such as people, goods or traffic network, from one location to another. On the one hand, flow maps can reduce visual clutter by merging single representations of movement. On the other hand, flow maps are also fast to produce and simple to understand. In this paper, we present a new method for the automatic generation of flow maps. Our method is based on a theoretically grounded physical system to describe the motion and forces of attraction and repulsion between data points. Additionally, support for an optional supervision of the graph layout is also available. Finally, the cost of our algorithm is evaluated and a comparison with existing implementations is provided. Results have shown a good balance between computational complexity and the visual quality of the generated maps.

Keywords

Flow maps, force-directed algorithm, geovisualization, graphical interaction

1 INTRODUCTION

Flow maps depict the movement of phenomena between geographic locations [25]. Phenomena can represent the movement in geographical space of both tangible (e.g. people, bank notes, and goods) and intangible objects (e.g. energy, ideas, and reputation). The links in a flow map, called flow lines, describe the movement of objects from one location to another. The way flow lines are aggregated and depicted is what makes a given flow map' algorithm unique.

The Sankey flow drawing technique [23] is an example of a method to aggregate flow lines. It describes the thickness of the aggregated flow lines to be proportional to the sum of the flows' magnitude they represent. Although the idea of defining the thickness to be proportional to the magnitude was intensively used by Matthew Sankey in 1898, it is known that he was not first to use it.

In this paper, we describe an automatic technique for the generation of natural and high visual quality flow maps using a force directed algorithm. On the one hand, it provides an alternative to already available tools for the generation of flow maps that shift the modelling effort to the user side [1]. On the other hand, our algorithm gives also the possibility of supervising the layout of the flow map during the creation process. Our algorithm not only shows a good computational complexity, but it also satisfies the following aesthetic criteria for flow maps [19, 27]:

C1. The use of smooth curves for aesthetic purposes;
C2. The possibility of aggregate flows, reducing the visual clutter;
C3. Emphasises the main branches of the flow tree; straight lines are correlated to target destinations with high magnitude;
C4. The target destinations represented by geometrical features such as circles are not overlapped with flow;
C5. The flow is crossing-free.

This paper is structured as follows. The next section surveys the relevant literature, both to see if similar studies have been done, and to provide the framework from which to evaluate the relevance and impact of this study. The third section of the study explains the proposed algorithm and covers all the implementation details. The fourth presents the benchmarks based on a test case and explores the meaning of this study in terms of visual appearance and performance. The last section wraps up possible extensibility.

2 RELATED WORK

In this section, we describe relevant techniques to the generation of flow maps and edge bundling algorithms.

 ISBN 978-80-86943-70-1

2.1 Flow map algorithms

The first flow map representation was created by Henry Harness in 1837 [21] and then popularised by Charles Minard around 1850 [22]. Minard is considered a pioneer in the use of graphics in engineering and statistics, mainly due to his map on the subject of Napoleon's disastrous Russian campaign of 1812 [16]. In 1987, Waldo Tobler designed the first software for an automatic generation of flow maps, called FlowMapper. This software draws flow lines as simple straight lines connecting both source and target points [26]. Although the biggest issue of FlowMapper is the visual clutter it produces, there are techniques that can be used to improve the final output. For example, we can sort flow lines by flow magnitude, use transparency, and apply personalised filters based on their magnitude, length, etc. To reduce the number of nodes, we can apply clustering methods, e.g. based on geographic proximity or political regions [7]. However, as we shall see, these tricks are not always an acceptable solution.

In 2005, Phan et al. [19] developed an innovative method to automatically generate flow maps. The method is characterised by its aggregation technique and curved edges. The authors defined a binary hierarchical clustering to formulate the layout in a simple and recursive manner. During the rendering phase, paths are drawn as cubic Catmull-Rom splines [8]. Then, in a post creation phase, users are given the possibility to alter the shape of the flow lines by moving their control points. The advantage of this algorithm is its computational complexity; a quadratic worst-case time. For example, on a 1.4-GHz laptop, the creation of a flow map has an order of magnitude of seconds. However, the algorithm has also its limitations. On the one hand, visual nodes are often moved from their original position if their proximity is too small, therefore losing their geographical reference. On the other hand, if there are too many target nodes in a small area, by forcing binary splits, it introduces too many extra routing nodes that leads to an unreadable clutter.

Verbeek et al. [27] introduced a method to overcome the aforementioned limitations by using directed angle-restricted Steiner trees of minimum length, also known as spiral trees. The creation of these spiral trees can be computed in $O(n \cdot \log(n))$ time, where n is the number of target nodes. The edges of the spiral trees are logarithmic spirals, implemented as cubic Hermite splines. Flow trees are designed to avoid crossing nodes, as well as user-specified obstacles. In order to straighten and smooth the tree, a cost function is defined. At each iteration, a new position of the intermediate nodes is computed and the cost function is calculated. Then they apply the steepest descent to minimise the global cost function, which is the sum of the cost function for each intermediate node. The layout is updated accordingly to the magnitude of the flows. Although the time complexity is not mentioned, it is expected to be $O(i \cdot m \cdot n)$, where i is the number of iterations, proportional to the number of target nodes n, and m is the set of all nodes. The time required to create a flow map is described as a 'couple of minutes' for large-scale flow maps (hundreds of target nodes) and less than a minute for localised flow maps (tens of target nodes). In addition, the flow tree has a unique topology on a given set of georeferred points. Hence, the same tree is used to represent all temporal datasets where only the magnitude varies. This bears some limitations since the magnitude is considered only after the tree generation.

Nocaj et al. [18] proposed an approach based on a new drawing style for the generation of flow maps called confluent spiral drawings. Confluent spirals consist of smooth drawings of each bundle in which edge directions are represented with smooth appearance. A sequence of spiral segments is used to represent an edge between the origin and the target node, and the vortex of each spiral corresponds to a target node. At the first step, edges are partitioned in $O(m \cdot log\triangle)$, where m is the number of edges and $\triangle$ is the maximum degree of a vertex. The overall runtime of the main step is $O(k \cdot n \cdot \triangle^2)$, where k is the number of candidate points over each spiral used as possible starting points for other spirals and n is the number of target nodes. The target nodes do not cross the flows if the edges are approximated with s segments. When an obstacle overlaps a branching point, it will be branched out in an earlier or later phase of the parent spiral to miss that obstacle. The computational time required is $O(n \cdot logn) + O(s \cdot log(n + s\triangle))$.

In 2013, a force directed algorithm [10] was introduced to automatically generate flow maps. The algorithm generates a set of intermediate nodes from the original flow map and then performs nodes merging using a force-directed strategy. The downsides of this approach are the high time complexity and the number of parameters, which consequently affect the final layout of the flow map. Moreover, it does not implement a smooth line model in its rendering phase. In this paper, we have significantly improved those aspects, as well as introduced novel features to support user supervision. A study on the influence of the number of intermediate nodes on the final result is presented as well.

2.2 Edge bundling techniques

Visual clutter is a common issue in the graph drawing domain for small as well as large graphs [17]. Force-directed algorithms [15] can be used to rearrange nodes' positions. In a force-directed algorithm the graph is represented as a physical system of particles with forces acting between them. At each iteration, the energy of the system changes. The algorithm halts when local minimum of the energy is found.

The combination of attractive forces on adjacent vertices, and repulsive forces on all vertices, was first introduced by Eades et al. [11]. A few years later, similar methods were presented as an extension to this idea. For example, Kamada and Kawai [14] introduced the idea of using only spring forces between all pairs of vertices, with ideal spring lengths equal to the vertices' graph-theoretic distance.

In recent years we have seen an increasing interest on force-directed algorithms, mainly in edge bundling. In edge bundling, the edges of a graph are bundled together if certain conditions are met. Holten et al. [13] presented a force-directed algorithm in which edges were modelled as flexible springs that can attract each other while node positions remain fixed. This algorithm was extended to separate opposite-direction bundles, emphasising the structure of the graph [24]. Cui et al. [9] described a mesh-based edge-clustering method for graphs. Control mesh generation techniques were used to capture the underlying edge patterns and to generate informative and less cluttered layouts. Ersoy et al. [12] created bundled layouts of general graphs, using skeletons as guidelines to bundle similar edges. Pupyrev et al. [20] proposed an edge routing algorithm based on ordered bundles. With this technique the edges are placed in parallel channels to avoid overlap.

One drawback of edge bundles is that they "hide" explicit node to node links. Hence, it is not easy to understand which are the nodes connected to a certain origin. Although, the main objective of these techniques is the reduction of the visual clutter, at the stage of the research, bundling methods cannot be used for the creation of Sankey flow maps; they do not accurately represent the total magnitudes flowing and they do not merge.

3 SYSTEM DESIGN

In this section, we describe the four components that characterise our application (see Figure 1). The first layer, entitled *Basic Structure Generation*, is responsible for the generation of the flow tree. Once the tree structure is generated, a second layer - *Flow Graph Layout* - is responsible for applying interaction forces to nodes and then for updating the tree structure accordingly. As described later, this phase is executed for a number of iterations. Then, the third layer - *Rendering* - depicts the flow map. Finally, the layer *Interaction* is designed to give users the possibility to modify the output by means of simple user's actions.

The algorithm receives as an input the geographic position of the origin, that is, the root node r of the tree t and n geographical destinations corresponding to the leaves $l_1, l_2, \ldots, l_n$, respectively with flow magnitudes $mgn(l_1), mgn(l_2), \ldots, mgn(l_n)$.

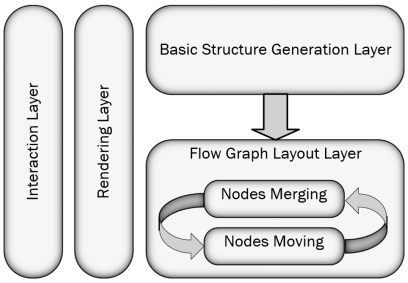


Figure 1: Overall system diagram.

3.1 Basic Structure Generation Layer

Initially each flow line, i.e., the line that connects the root r with a leaf node l_i, is a straight line. The first step is to divide each flow line in sub-segments of length d composed by the intermediate nodes $m_{i,j}$. The index i identifies the leaf node l_i of the original segment. This index takes into account the position of the leaves in clockwise order. The sorting operation required to obtain such indexes has a computational cost of $O(n \cdot \log(n))$. The index j identifies the order of the intermediate nodes associated to l_i. At a distance d from the root r, there will be the intermediate nodes $m_{1,1}, m_{2,1}, \ldots, m_{n,1}$. At a distance $2d$, the set of nodes is $m_{1,2}, m_{2,2}, \ldots, m_{n,2}$, and so on (see Figure 2). Therefore, the number of intermediate nodes $m_{i,j}$ is proportional to the distance between a leaf l_i and its root r. Each intermediate node $m_{i,j}$ is tagged with the following information:

- the parent node; initialised as $pred(m_{i,j}) = m_{i,j-1}$.
- the child nodes; where $next(m_{i,j}) = \{m_{i,j+1}\}$ is the initial set.
- the interacting nodes; a set containing the surrounding nodes. $int(m_{i,j}) = \{m_{i+1,j}, m_{i-1,j}\}$ is the initial set. Leaves cannot be interacting nodes.
- the magnitude: $mgn(m_{i,j}) = mgn(l_i)$.

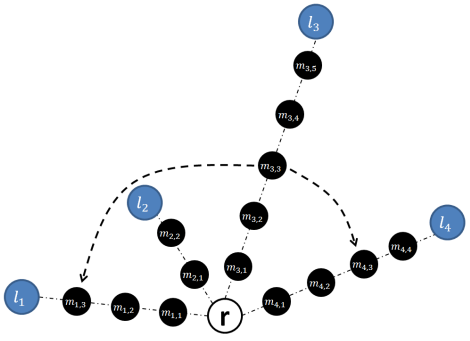


Figure 2: Nodes representation: $m_{3,3}$, $mgn(m_{3,3}) = mgn(l_3)$, $pred(m_{3,3}) = m_{3,2}$, $next(m_{3,3}) = \{m_{3,4}\}$ and $int(m_{3,3}) = \{m_{1,3}, m_{4,3}\}$.

When a node $m_{i,j}$ does not have as interacting node the node $m_{i-1,j}$ (due to the fact that the distance from $m_{i,j}$ to r is greater than the distance from l_{i-1} to r), it will be substituted with the first of the previous ones, as shown in Figure 2. The same applies to the absence of the node $m_{i+1,j}$. Consequences are explained in Section 4.1.

3.2 Flow Graph Layout Layer

This layer aims at computing the flow graph layout. Figure 3 illustrates the evolution of a tree structure during the execution of the algorithm. The two steps that define this layer are executed in sequence, and for a number of iterations.

3.2.1 Nodes merging phase

This phase was designed to update the structure of the tree t, which becomes obsolete after the merging procedure. The following pseudo-code describes the merging steps:

for each intermediate node $m_{i,j}$ **do**
  for each $s \in int(m_{i,j})$ **do**
    if $samePosition(m_{i,j},s)$ & $Pred(m_{i,j}) \equiv Pred(s)$ **then**
      $createNode(s,m_{i,j})$
      $removeNode(s)$
      $removeNode(m_{i,j})$
    end if
  end for
end for

The function *samePosition* requires a threshold d_s to define whenever two positions are equivalent. The existence of a common parent for s and $m_{i,j}$ is a necessary condition to maintain the structure as a tree and not only as a directed acyclic graph. The function *createNode* inserts a new node. The new node shares the parent node of s and $m_{i,j}$; the child nodes of s and $m_{i,j}$; has as interaction nodes the interaction nodes of s and $m_{i,j}$; has as magnitude the sum of the magnitudes of s and $m_{i,j}$; and has as position the middle position between s and $m_{i,j}$. The function $removeNode(k)$ deletes the intermediate node k from the system. The computational complexity of this phase is $O(m)$ in each iteration.

3.2.2 Nodes moving phase

This phase aims at readjusting the position of intermediate nodes. The behaviour of each intermediate node depends on two forces: the attractive and the stress force. The attractive force is calculated only between the node and its interactive nodes, within a certain distance d_a. The attractive force F_a, is given by the equation:

$$F_a(m_{i,j}) = \sum_{s \in Int_{m_{i,j}}} \frac{mgn(s)}{mgn(s)+mgn(m_{i,j})} \frac{1}{\left\|m_{i,j}-s\right\|} \hat{(m_{i,j}-s)} \quad (1)$$

The notation $\hat{A}$ and $\|A\|$ gives respectively the unit vector and the norm of A. The factor $\hat{(m_{i,j}-s)}$ gives the direction of the force vector and $\frac{1}{\left\|m_{i,j}-s\right\|}$ the force length; the closer the two nodes are, the higher is the force between them. $mgn(s)$ and $mgn(m_{i,j})$ are respectively the magnitude of s and $m_{i,j}$. Without the factor $\frac{mgn(s)}{mgn(s)+mgn(m_{i,j})}$ the formula takes into account the distance of the interacting nodes but not their magnitude. We want nodes of smaller magnitudes to be attracted by nodes of higher magnitude. Moreover, we want flows with higher magnitude to have straighter shapes than the ones with lower magnitude.

In addition to the attractive force, a stress force F_s is applied to each intermediate node, to keep a 'middle-aligned' position from the parent node, as well as from the child nodes.

$$F_s(m_{i,j}) = (prev(m_{i,j}) - m_{i,j}) + \sum_{s \in next_{m_{i,j}}} \frac{mgn(s)}{mgn(m_{i,j})}(m_{i,j}-s) \quad (2)$$

The condition defined in Equation 2 ensures that nodes move always towards children of higher magnitude. $mgn(m_{i,j})$ is the magnitude of the current node, that is, the sum of the magnitude of all children. $mgn(s)$ is the magnitude of a child node. In order to avoid oscillations, the stress force is applied only if the force is greater than a threshold t: $F_s(m_{i,j}) > t$. The final formula to compute the force corresponds to the sum of the stress and the attractive forces:

$$F_{final}(m_{i,j}) = F_a(m_{i,j}) + k_s \cdot F_s(m_{i,j}) \quad (3)$$

where the constant k_s is the oscillation that defines the stress force. In this context, the concept of force is the one of displacement applied to a node.

The total force of the system is equal to the scalar sum of all forces applied to the intermediate nodes. After each iteration the total force decreases, firstly because the number of nodes is reduced, and then because the interacting nodes become out of range (i.e. when the distance from their associated node is greater than d_a).

The tree structure is marked as ready once a stable total force is reached. Unfortunately, the tree structure does not take into consideration the overlap of nodes, one of the criteria that this algorithm aims to achieve. Hence, the objective of the next step is to apply a repulsive force F_r to intermediate nodes within a certain distance d_r from the leaf nodes. The displacement is applied only at the end because otherwise it would affect the evolution of the tree t.

$$F_r(m_{i,j}) = \sum_{l \in leaves} \frac{-1}{\left\|m_{i,j}-l\right\|} \hat{(m_{i,j}-l)} \quad (4)$$

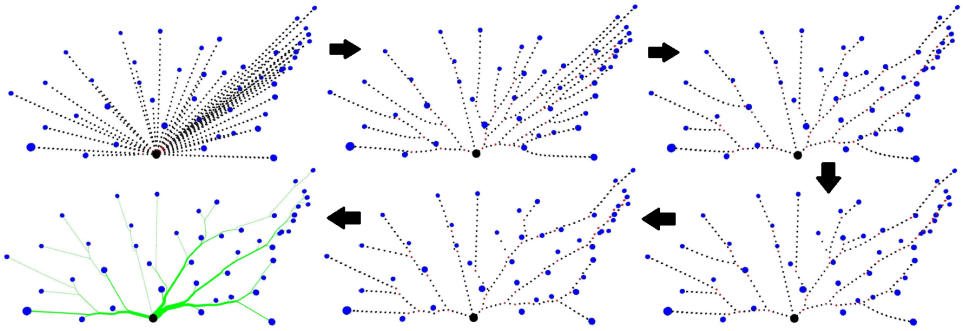


Figure 3: Tree structure during the generation of the flow map.

The repulsive force is equal to the opposite of the attractive force without the factor that takes into account the magnitudes. Additionally, we add the stress force to guarantee that the position of children nodes is updated properly once the node location changes (see Figure 4).

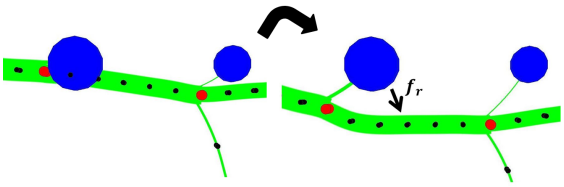


Figure 4: (left) Overlapping between a flow line and a leaf node. (right) Repulsive and stress forces are applied to avoid overlapping.

In pseudo-code the operations of this phase are described as follows:

for each intermediate node $m_{i,j}$ **do**
  if $SystemIsStable == FALSE$ **then**
    $displ(m_{i,j}) = F_{final}(m_{i,j})$
  else
    $displ(m_{i,j}) = F_r(m_{i,j}) + k_s \cdot F_s(m_{i,j})$
  end if
end for
for each intermediate node $m_{i,j}$ **do**
  $position(m_{i,j}) += displ(m_{i,j})$
end for
$SystemIsStable = checkStability()$

In summary, the execution of the algorithm runs in two steps. First, Equation 3 is applied until the total force of the system is stable. Then the attractive force is substituted with the repulsive force to avoid overlapping with destination nodes. When the system becomes stable for the second time the algorithm stops. At the end, the intermediate nodes' position is updated.

3.3 Rendering Layer

The classes responsible for rendering the maps on the screen are defined in the Rendering layer. The 3D rendering library used is JOGL, a binding of OpenGL for Java, which has been released by Oracle for Windows, Solaris, Linux and Mac OS platforms. Maps can be visualised both in 3D and 2D.

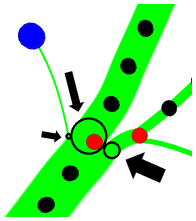


Figure 5: Splits of flow lines. Unfilled circles shows the starting point of the child's spline.

For aesthetic purposes, curves were preferred over straight lines during the rendering phase [6]. Xu et al. published a study to empirically evaluate their effectiveness on common graph-related tasks [28]. In our implementation, we used natural Cubic Splines for the depiction of the flow lines. We decided for this kind of curve representation because it interpolates the intermediate nodes and its design dynamics are intuitive. The line width is proportional to the magnitude of the intermediate nodes that compose the flow. Moreover, for each split the starting point of the child's spline is shifted by a distance proportional to their width, in a direction perpendicular to the edge' vector (see Figure 5).

Additionally, leaf nodes are marked as circles whose size is proportional to their magnitude. It is possible to visualise the tree structure as well; intermediate nodes are depicted in red if they have more than one child node and black otherwise.

3.4 Interaction Layer

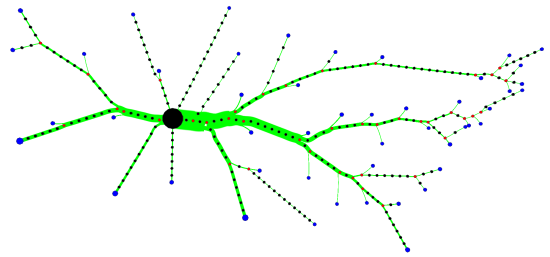
The interaction layer enables the possibility of interacting with the structure of the flow graph tree, and consequently can improve its visual quality. At the current state of the research, automatic methods for the generation of flow maps do not support supervised layouts before concluding the generation process. Manual updates are possible once the layout is generated.

The interaction layer supports two scenarios: the automatic generation of a flow map and the supervised step-

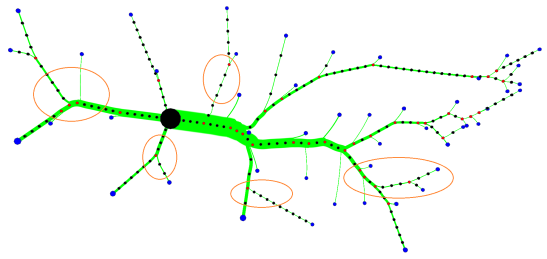
by-step layout generation. The latter option gives users additional control over the final layout. Hence, a given area can be progressively refined, for example, by moving certain intermediate nodes to a different location. After applied the translation force, the system takes it into account and recomputes the new flow graph tree.

The user interface has three input components to control the generation process. A button *next* to compute the *node merging phase* and the *node moving phase* for the next step. Since the number of iterations can be in the order of ~ 1000, the user has also the possibility to run it in batches. At any moment, the user has the possibility to manually modify the attractive, stress and repulsive forces and decide when its time to stop the algorithm execution.

As described in Section 3.2, the algorithm is responsible for the management of all the forces involved in the system.



(a) Automatically generated flow map.



(b) Flow map generated with the supervision of the user.

Figure 6: Flow maps illustrating the migration from California between 1995-2000. Intermediate nodes are visible only during the supervised mode. The ovals identify the areas where the user requested the aggregation of flows.

The use of the supervised mode is useful when a flow map is composed by a large number of branches. In such scenario, the user can 'drag & drop' intermediate nodes with the goal of activating new attractive forces to reduce the number of branches of the tree structure. The Figure 6 shows a geographical dataset depicted with and without the user supervision.

4 EVALUATION OF THE ALGORITHM

In this section, we compare our algorithm to existing implementations. We also provide an evaluation of its computational cost. Results have shown a good balance between computational complexity and the visual quality of the generated maps.

4.1 Comparison with existing algorithms

Table 1 provides a description of the most relevant methods for the automatic generation of flow maps accordingly to the aesthetic criteria defined in Section 1. The Y/N letters stand for Yes/No and the properties marked with a star have limitations that are also explained.

Methods	C1/C2	C3	C4	C5	Time Complexity
Flow Map Layout	Y	Y*	N	N	$O(n^2)$
Flow Map Layout via Spiral Tree	Y	Y*	Y	Y	$O(n \cdot log(n)) + O(i \cdot m \cdot n)$
Confluent Spiral Drawing	Y	N*	Y	Y	$O(n \cdot log(n)) + O(k \cdot n \cdot \triangle) + O(s \cdot log(n + s\triangle))$
Force Directed Flow Map Layout	Y	Y	Y*	Y*	$O(n \cdot log(n)) + O(i \cdot m \cdot n)$

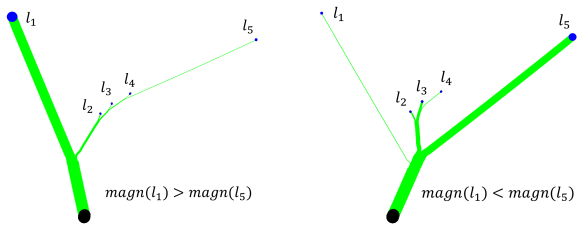
Table 1: Table for the comparison of the algorithms for automatically generated flow maps.

C1. All methods evaluated in this section use smooth lines. The flow map layout algorithm [19] uses Catmul-Rom splines, Flow Map Layout via Spiral Tree [27] uses cubic Hermite splines and Confluent Spiral Drawing [18] uses logarithmic spirals. Our algorithm uses a natural Cubic Spline to represent flow lines.

C2. The first two methods compute the aggregation of flow lines through the creation of binary splits. Unlike the first, the second method can recursively perform binary splits that are then merged if the distance meets a certain threshold. Confluent Spiral Drawing and our algorithm do not impose such limitation.

C3. The criterion states that the flow magnitude should affect the layout of the generated map. However, such condition is not completely satisfied by the first three algorithms. In the first two algorithms only the position of the intermediate nodes is affected meanwhile the structure of the tree remain unmodified. The authors justify such restriction with the claim that it helps the comparison across different time periods. In the third algorithm the magnitude is not taken into consideration. Although the authors claimed the possibility to use other attributes instead of only the distance from the root node, their paper not goes into details describing this aspect. Nevertheless, it can be useful to have

a dynamic tree typology to give emphasis to flow lines with greater flow magnitude.



(a) (b)

Figure 7: Representation of a flow maps with different magnitudes

As showed in Figure 7, our method can depict a flow tree with a structure specifically designed to account for the magnitude of each destination node, that is, it takes into consideration the magnitude factor in the force behaviours.

C4. The leaf nodes are not overlapped for all the methods except the first; a buffer is used to distance the flow lines from the nodes. In our algorithm if the number of intermediate nodes is not sufficient some overlapping can occur - i.e if the d_r is less than $d/2$.

C5. In our algorithm the overlap of flow lines depends on the criteria used to define the interacting nodes. If the rule is to assign for each $m_{i,j}$ the nodes $m_{i-1,j}$ and $m_{i+1,j}$ (if they exist) as interacting nodes, the tree is crossing-free. On the other hand, if we extend the rule as described in section 3.1 (for example the node $m_{1,3} \in int(m_{3,3})$ in Figure 2), the overlapping is permitted but the quality of the generated output is improved. In Figure 8 is possible to assess the benefits of this decision. However, by increasing the number of intermediate nodes we can reduce considerably the possibility of such scenario.

Figure 9 shows the comparison of flow maps depicted with Flow Map Layout, Flow Map Layout via Spiral Tree, Confluent Spiral Drawing, as well as with our algorithm. The output of the first algorithm contains many crossings, the grouping of nodes is somewhat unnatural, and the edges are often difficult to follow. The other outputs can be considered aesthetically pleasing, although no methodology exists to evaluate flow maps generated automatically.

4.2 Evaluation of the computational complexity and visual quality

To decide the number of intermediate nodes to generate, we need to set the distance d between the intermediate nodes. We define the parameter f_n to be the maximum number of intermediate nodes in a flow line. Then we assign $d = d_{max}/(f_n + 1)$, where d_{max} is the length of the longest flow line. This parameter consequently affects the performance and visual quality of

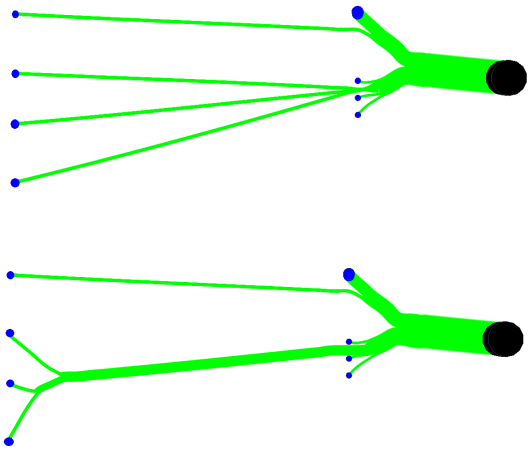


Figure 8: This example highlights a weak case in terms of visual quality and how it is solved. A flow map where the absence of overlapping is guaranteed (top), flow map with the possibility of overlapping but with better visual quality (bottom).

the flow map. Figure 10 compares the number of nodes, time and iterations for a flow map that depicts the top 30 exports of Whisky from the UK. Let $f_n = 30$ and $m = 338$ be the initial number of intermediate nodes, then the algorithm stops after 2,51 seconds and 1597 iterations. With $f_n = 60$ and $m = 695$, the algorithm halts after 9,46 seconds and 2487 iterations. For $f_n = 100$ and $m = 1177$, the algorithm stops after 23,1 seconds with a total of 4240 iterations. Note that a small number of intermediate nodes does not guarantee that all flows are cross-free or the absence of crossing with the leaf nodes, see respectively the second and first map in Figure 10.

The plot depicting the total force illustrates when the algorithm finds a local minimum of the total force and when it applies the repulsive force instead of the attractive force to its nodes; for each of 3 cases the force has a pick and then it decreases until a minimum is reached. It is possible to notice that the convergence is fast using the total force of the system as cost value.

In the plot depicting time, when the stress repulsive force is introduced to overcome the overlapping of destination nodes, the function increases the time; in the first step the time complexity is in the order of $O(i \cdot n)$, meanwhile in the second step the destination nodes must be considered, making the time complexity in the order of $O(i \cdot m \cdot n)$. Our algorithm performs a flow map with intermediate nodes in the order of one thousand in less than a minute.

In addition to f_n and distance d between intermediate nodes, we have the following parameters: the threshold d_s that defines when two positions are equivalent; the

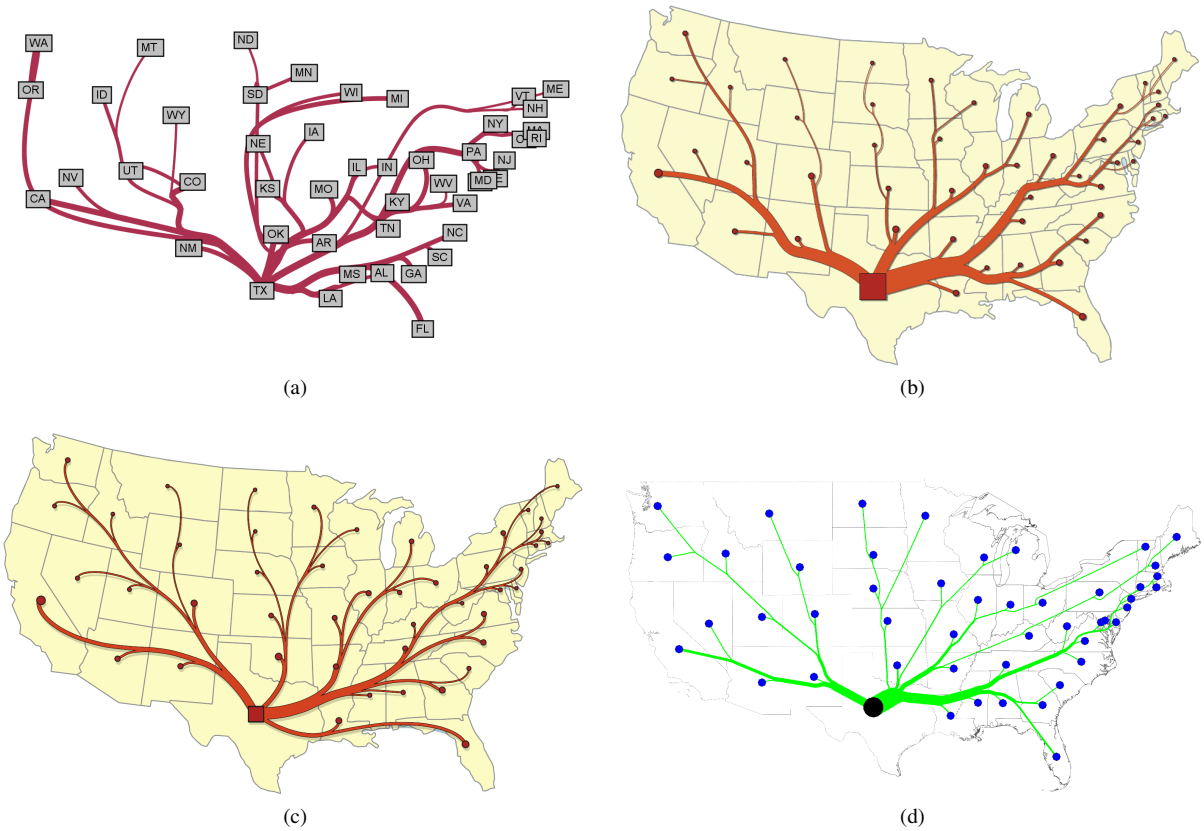


Figure 9: Maps illustrating migration from Texas 1995-2000. The output of [19] (a), the output of [27] (b), the output of [18] (c) and the output of our algorithm (d).

distance d_a for the attractive force; the distance d_r for the repulsive force; and the constant oscillation k_s that varies between 0 to 1. After performing several tests we concluded empirically that 0.1 is an optimal value.

Figure 11 depicts the effects of varying the main parameters of the algorithm. On the top there are three flow maps varying the number of intermediate nodes, on the left the factor f_n is 12, in the center f_n is 50, and on the right the value is 100.

We observed that in situations where the number of nodes is low, the merging of flow lines is reduced and the result is often not satisfactory. We have also noticed that after a certain threshold the algorithm produces no significant improvements.

On the bottom we see what happens if the factor that takes into account the magnitude is removed from Formula 1 (center image) or from Formula 2 (right image).

The first case generates sparser results. In the former case, the position of each split does not follow the direction of the lines (for example, the main branch on the right of the origin). In the bottom left the range for the attractive force is reduced (low d_a), the number of branches is increased mainly near the origin.

The datasets used in this paper can be retrieved online from: the Scotch Whisky Association in Edinburgh [4], Statistics Norway [5] and U.S. Census Bureau, County-to-County Migration Flows [2].

All flow maps depicted in this paper were computed and visualised on a Intel laptop with 1.64 GHz and 2 GB of RAM. The algorithm was initially developed using a 2D Java canvas, but was later ported to 3D and integrated in NASA WorldWind [3]. Hence, both 2D and 3D views are supported.

5 CONCLUSION AND FUTURE WORK

This paper describes a force-directed algorithm for the automatic generation of flow maps. The algorithm for computing the flow map iteratively minimizes the energy of a system composed by a set of forces that characterise a well-drawn flow map.

The main aspect is that the flow tree, as well as the flow lines, are mainly based on the magnitude of the destinations. The second aspect is the possibility to supervise the output during each iteration. Even if the visual quality of our algorithm was not proved to be the best compared with the previous work, it depicts good quality flow maps. Moreover, the method is of easier implementation, and the time requested for the execution is

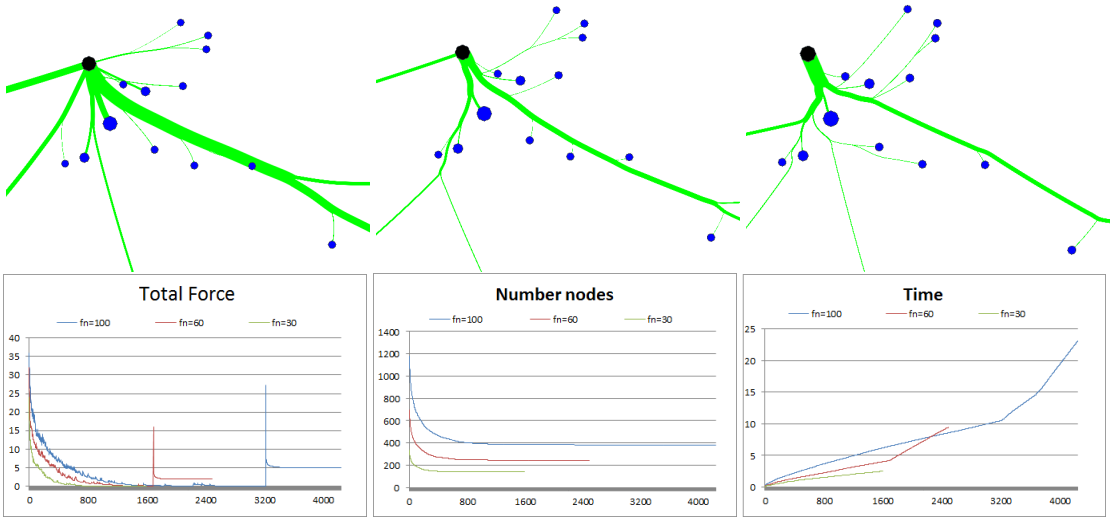
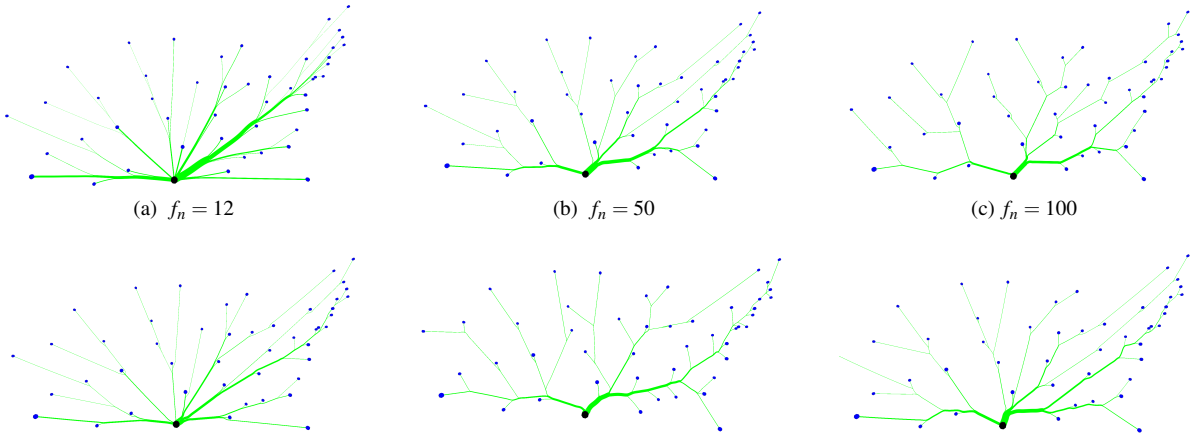


Figure 10: On top: flow map with $f_n = 30$ (left), flow map with $f_n = 60$ (centre), flow map with $f_n = 100$ (right). On bottom: the x axis represents the number of iterations and the y axis the total force of the system (left plot), the number of nodes (central plot), and the execution time (right plot).



(a) $f_n = 12$ (b) $f_n = 50$ (c) $f_n = 100$

(d) low d_a (e) the magnitude factor is removed from equation 1 (f) the magnitude factor is removed from equation 2

Figure 11: Various results of our algorithm with different parameters.

in the order of seconds for flow maps of thousand of intermediate nodes.

As a further improvement, we could use a quadtree to store the position of each node and to reduce the time needed to avoid crossings between leaf nodes and flow lines. The algorithm could also be extended to support multi-origin representations. The use of a 3rd dimension can be another challenging direction for future work.

6 ACKNOWLEDGEMENTS

This research has been supported by the European Commission (EC) under the projects i-SCOPE (Grant Agreement N. 297284), SUNSHINE (Grant Agreement N. 325161) and GEPSUS. The project GEPSUS is funded by NATO-OTAN (North Atlantic Treaty Organization) within the Science for Peace and Security Programme. The authors are solely responsible for the content of this paper. It does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of information contained herein.

7 REFERENCES

[1] Adobe illustrator. `http://www.adobe.com/it/products/illustrator.html`. Accessed: 2014-04-24.

[2] County-to-county migration flows. `http://www.census.gov/population/www/cen2000/ctytoctyflow/index.html`. Accessed: 2014-04-24.

[3] Nasa worldwind. `http://worldwind.arc.nasa.gov/java/`. Accessed: 2014-04-24.

[4] Statistical report 2012. http://www.scotch-whisky.org.uk/media/62024/2012_statistical_report.pdf. Accessed: 2014-04-24.

[5] Statistics norway. http://www.ssb.no. Accessed: 2014-04-24.

[6] M. Bar and M. Neta. Humans prefer curved visual objects. *Psychological science*, 17(8):645–648, 2006.

[7] I. Boyandin, E. Bertini, and D. Lalanne. Using flow maps to explore migrations over time. In *Geospatial Visual Analytics Workshop in conjunction with The 13th AGILE International Conference on Geographic Information Science*, volume 2, 2010.

[8] E. Catmull and R. Rom. A class of local interpolating splines. *Computer aided geometric design*, 74:317–326, 1974.

[9] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1277–1284, 2008.

[10] A. Debiasi, B. Simoes, and R. De Amicis. Force directed flow map layout. In *5th International Conference on Information Visualization Theory and Applications*, pages 170–177. SCITEPRESS, 2014.

[11] P. Eades. A heuristics for graph drawing. *Congressus numerantium*, 42:146–160, 1984.

[12] O. Ersoy, C. Hurter, F. V. Paulovich, G. Cantareiro, and A. Telea. Skeleton-based edge bundling for graph visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2364–2373, 2011.

[13] D. Holten and J. J. Van Wijk. Force-directed edge bundling for graph visualization. In *Computer Graphics Forum*, volume 28, pages 983–990. Wiley Online Library, 2009.

[14] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989.

[15] M. Kaufmann and D. Wagner. *Drawing graphs: methods and models*, volume 2025. Springer, 2001.

[16] C. J. Minard. *Carte figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812-1813*. Graphics Press., 1869.

[17] T. Munzner. H3: Laying out large directed graphs in 3d hyperbolic space. In *Information Visualization, 1997. Proceedings., IEEE Symposium on*, pages 2–10. IEEE, 1997.

[18] A. Nocaj and U. Brandes. Stub bundling and confluent spirals for geographic networks. In *Graph Drawing*, pages 388–399. Springer, 2013.

[19] D. Phan, L. Xiao, R. Yeh, and P. Hanrahan. Flow map layout. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, pages 219–224. IEEE, 2005.

[20] S. Pupyrev, L. Nachmanson, S. Bereg, and A. E. Holroyd. Edge routing with ordered bundles. In *Graph Drawing*, pages 136–147. Springer, 2012.

[21] A. H. Robinson. The 1837 maps of henry drury harness. *The Geographical Journal*, 121(4):pp. 440–450, 1955.

[22] A. H. Robinson. *Early thematic mapping in the history of cartography*. University of Chicago Press Chicago, 1982.

[23] M. Schmidt. The sankey diagram in energy and material flow management. *Journal of Industrial Ecology*, 12(1):82–94, 2008.

[24] D. Selassie, B. Heller, and J. Heer. Divided edge bundling for directional network data. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2354–2363, 2011.

[25] T. A. Slocum, R. B. McMaster, F. C. Kessler, and H. H. Howard. *Thematic cartography and geovisualization*. Pearson Prentice Hall Upper Saddle River, NJ, 2009.

[26] W. R. Tobler. Experiments in migration mapping by computer. *The American Cartographer*, 14(2):155–163, 1987.

[27] K. Verbeek, K. Buchin, and B. Speckmann. Flow map layout via spiral trees. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2536, 2011.

[28] K. Xu, C. Rooney, P. Passmore, D.-H. Ham, and P. H. Nguyen. A user study on curved edges in graph visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2449–2456, 2012.