Flow Map Layout via Spiral Trees

Kevin Verbeek, Kevin Buchin, and Bettina Speckmann

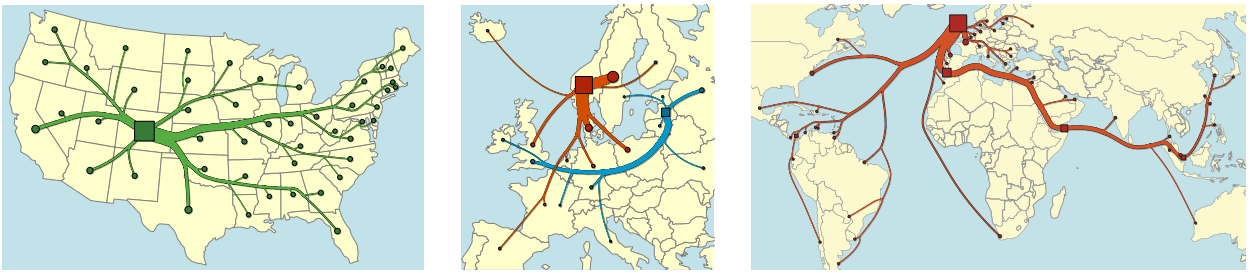

Fig. 1. Flow maps: Migration from Colorado, migration from Norway and Latvia, whisky exports from Scotland.

Abstract—Flow maps are thematic maps that visualize the movement of objects, such as people or goods, between geographic regions. One or more sources are connected to several targets by lines whose thickness corresponds to the amount of flow between a source and a target. Good flow maps reduce visual clutter by merging (bundling) lines smoothly and by avoiding self-intersections. Most flow maps are still drawn by hand and only few automated methods exist. Some of the known algorithms do not support edge-bundling and those that do, cannot guarantee crossing-free flows. We present a new algorithmic method that uses edge-bundling and computes crossing-free flows of high visual quality. Our method is based on so-called *spiral trees*, a novel type of Steiner tree which uses logarithmic spirals. Spiral trees naturally induce a clustering on the targets and smoothly bundle lines. Our flows can also avoid obstacles, such as map features, region outlines, or even the targets. We demonstrate our approach with extensive experiments.

Index Terms—Flow maps, Automated Cartography, Spiral Trees.

1 Introduction

Flow maps are thematic maps that visualize the movement of objects, such as people or goods, between geographic regions [6, 20]. So called *distributive flow maps* are used to depict quantitative data, for example, the amount of wine exported by France or the magnitude of migration between countries. One or more sources are connected to several targets by lines whose thickness corresponds to the amount of flow between a source and a target.

A (distributive) flow map typically consists of one or more *flow trees* which are drawn atop a base map. A flow tree is a single-source flow, that is, it connects a single source (the root) to several targets (the leaves). The widths of the flow lines of a flow tree are scaled proportionally (linearly) to the values they represent. When a flow line (trunk) separates into several smaller lines (branches) the width of the branches should add up to the width of the trunk [6]. Flow trees sometimes follow the actual routes of the movement they depict. When this is not possible or not desirable they still give an overall impression of the distribution pattern and show trends.

The first flow map was created by Henry Drury Harness in 1837. Shortly after, Charles Joseph Minard began to create flow maps depicting mostly economic topics. His sophisticated designs were instrumental in popularizing flow maps. Despite their long history and popularity most flow maps today are still drawn by hand and few automated methods exist. We discuss these in Section 2.

Quality criteria. Good flow maps share some common properties. They reduce visual clutter by bundling lines as smoothly and frequently as possible. They also strive to avoid crossings between lines. Specifically, every flow tree is plane, that is, crossing free. A good flow tree avoids its own nodes, that is, all branches have a minimum length so that their widths can be interpreted. In addition, the main branches of a flow tree grow as straight and smooth as possible, to make it easy to follow them. When a flow map consists of several flow trees then the crossings between different trees should be minimized [6]. Thin branches should be drawn atop thick ones while avoiding a "weaving" effect. Furthermore, flow maps often try to avoid covering important map features with flow trees to aid recognizability.

Flow maps that depict trade often route edges along actual shipping routes. In that case a moderate distortion of the underlying geography is admissible. In contrast, flow maps that depict abstract data or data which is not linked to specific routes, such as migration or internet traffic, do not distort the geography of the underlying map [20].

Results and organization. We present an automated approach to generate high-quality flow maps. Specifically, we describe an algorithm that bundles edges and produces crossing free flow trees that have a natural and smooth appearance. The flow trees generated by our algorithm are able to avoid their own nodes, as well as user specified obstacles. We do not distort the underlying geographic base map, since such a distortion may be confusing [20]. Our algorithm can merge internal nodes that are too close and so create vertices of degree higher than three. This alleviates clutter in congested areas. Our flow trees use a unique topology on a given set of input points and therefore lend themselves to animations of changing data sets over time.

Our method is based on so-called *spiral trees* [3], a novel type of *Steiner tree* [12] which use logarithmic spirals. A (geometric) Steiner tree for a set P of points in the plane is a graph of minimum length that connects all points in P either directly or via additional so-called Steiner points. It can be shown that such a graph is always a crossing-free tree. Steiner trees naturally cluster the input points, but they have angles of 120° at every internal node and hence are quite far removed from the smooth appearance of hand-drawn flow maps. The edges of spiral trees, on the other hand, obey a certain restriction on the angle they form with the root. As a consequence, spiral trees do not only induce a natural clustering on the leaves but also smoothly bundle lines.

- *Kevin Verbeek is with TU Eindhoven, E-mail: k.a.b.verbeek@tue.nl.*
- *Kevin Buchin is with TU Eindhoven, E-mail: k.a.buchin@tue.nl.*
- *Bettina Speckmann is with TU Eindhoven, E-mail: speckman@win.tue.nl.*

Manuscript received 31 March 2011; accepted 1 August 2011; posted online 23 October 2011; mailed on 14 October 2011.
For information on obtaining reprints of this article, please send email to: tvcg@computer.org.

Fig. 2. Maps illustrating migration from California 1995–2000. Top: flow maps, Tobler [1, 22] (arrows of varying width), Phan *et al.* [16] (edge-bundeling with crossings), and our output. Bottom: subgraphs of the bundled complete migration graph, Cui *et al.* [5] and Holten & van Wijk [11].

We describe an algorithm that thickens and smoothes a given spiral tree while avoiding obstacles. Our algorithm minimizes a set of cost functions that capture the quality criteria for flow trees.

The remainder of the paper is organized as follows. Section 2 discusses related work and visually compares our flow maps to those produced by other automated methods. Section 3 gives a brief introduction to spiral trees. Section 4 presents our algorithm to compute flow trees. Since our flow trees are based on spiral trees and hence use logarithmic spirals for their edges, we define a novel type of spline, the *spiral spline*, to draw them effectively. Spiral splines are in essence cubic Hermite splines where we use logarithmic spirals to compute the tangents. Section 5 shows and discusses results of our algorithm, also in comparison with previous work. Section 6 discusses various extensions to our automatic method: flow maps that support user specified clusters on the leaves of the tree, flow maps that contain more than one flow tree, and flow maps that route edges via waypoints or along shipping routes. Section 7 gives some technical details about our implementation. In Section 8 we close with a discussion of our work.

2 Related work

Since their introduction in the mid-1800s, flow maps have been a widely used type of thematic map [17]. Textbooks on thematic cartography provide design rules for flow maps [6, 20]. The first known systems for the automated creation of flow maps were developed in the 1980s [1, 8, 22]. These systems do not merge flow lines and hence the resulting maps suffer from visual clutter (see Fig. 2, top-left). In 2005 Phan *et al.* [16] presented an algorithm, based on hierarchical clustering of the leaves, which creates single-source flow maps with bundled edges (see Fig. 2, top-middle). This algorithm uses an iterative ad-hoc method to route edges and hence is often unable to avoid crossings. A second effect of this method is that flows are often routed along counterintuitive routes. The quality of the maps can be improved by moving the leaves, which, however, is considered to be confusing for users according to cartography textbooks [20]. In contrast our method merges flow lines and avoids unnecessary crossings (see Fig. 2, top-right).

Flow maps are effective if the number of origin-destination pairs is linear in the number of origins and destinations. This is the scenario on which we focus in this paper. If flows between all origins and all destinations need to be shown then the number of origin-destination pairs is quadratic. There are some recent papers that therefore explore alternative ways to visualize flows, by using multi-view displays [9], animations over time [2], or mapping techniques close to treemaps [23]. By interpreting origins and destinations as nodes and flows as (weighted) edges, techniques from graph and network visualization can be used to visualize flows. Cox *et al.* [4] and Munzner *et al.* [15] use 3D maps to visualize network structure. Their methods, however, do not use edge-bundling. In contrast, Holten [10], Cui *et al.* [5], and Holten & van Wijk [11] present methods to visually bundle edges in complete and dense graphs. Subgraphs of their bundled representations can be interpreted as flow maps (see Fig. 2, bottom). These subgraphs, however, are not necessarily crossing-free, or even trees, and do not merge as quickly and smoothly as hand-drawn flow maps.

Duncan *et al.* [7] describe a model for drawing with fat edges. In principle we could use their method to thicken spiral trees, but the resulting flow trees are not smooth, do not obey angle restrictions, and do not emphasize the main branches of the tree. Finally, Krozel *et al.* [13] study algorithms for turn-constrained routing with thick edges in the context of air traffic control. Their paths need to avoid obstacles (bad weather systems) and arrive at a single target (the airport). The union of consecutive paths bears some similarity with flow maps, although it is not necessarily crossing-free or a tree.

3 Spiral trees

In this section we give a brief introduction to spiral trees, for additional details please see our companion paper [3]. As mentioned above, spiral trees are a novel type of Steiner tree. To create trees with the smoothly merging lines of hand-drawn flow maps, we impose a restriction on the angle the edges form with the root. Specifically, we use a *restricting angle* $\alpha < \pi/2$ to control the direction of the arcs of a geometric directed Steiner tree T. Consider a point p on an edge e from a leaf to the root r (see Fig. 3). Let γ be the angle between the vector from p to the root r and the tangent vector of e at p. We require that $\gamma \leq \alpha$ for all points p on T. We refer to a directed Steiner tree that obeys this angle restriction as *directed angle-restricted Steiner tree*. Here and in the remainder of the paper it is convenient to conceptually direct trees from the leaves to the root.

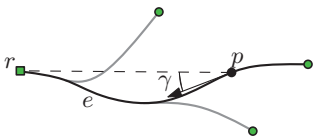

Fig. 3. The angle restriction.

The edges of an optimal (that is, shortest) directed angle-restricted Steiner tree T consist of straight line segments and parts of *logarithmic spirals*. For $-\pi/2 < \beta < \pi/2$ the β-spiral through a point p can be described by the following parametric equation in polar coordinates, where $p = (R, \phi)$: $R(t) = Re^{-t}$ and $\phi(t) = \phi + \tan(\beta)t$. We call β the *angle* of (a segment of) a β-spiral. A spiral tree with restricting

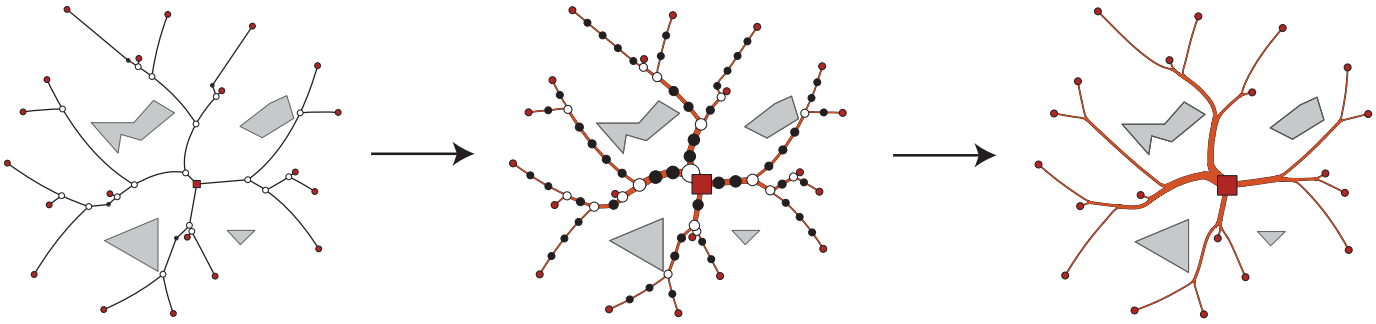

Fig. 4. Starting with a thin spiral tree (left), we thicken and further subdivide the edges (middle), and finally minimize the cost function (right, drawn without internal nodes). The trees consist of a root node (square), leaf nodes (red), join nodes (white), and subdivision nodes (black).

angle α is a directed angle-restricted Steiner tree whose edges consist only of segments of α-spirals and $(-\alpha)$-spirals. To compute flow trees we use $\alpha = 25°$, or $\alpha = 35°$ in the presence of large obstacles.

In [3] we prove that it is NP-hard to compute optimal spiral trees. We also present a 2-approximation algorithm. This algorithm can be extended to avoid obstacles, although we then cannot guarantee the approximation factor anymore. The resulting spiral trees, however, are of high visual quality (see Fig. 4 left). We use obstacles to cover all leaves and so force the tree to avoid its own nodes.

4 Computing Flow Trees

In this section we describe our algorithm to compute a single flow tree T. Our input is a point r, the root of T, and n points $l_1, \ldots, l_n$, the leaves of T. For every leaf l_i we are also given a weight t_i which denotes the amount of flow from the root to leaf l_i. The weight t_i determines the thickness of the flow tree at leaf l_i. We assume that these weights have already been scaled appropriately, using linear scaling as advocated by cartographers [6]. To simplify the discussion we further assume that the root r is located at the origin.

A good flow tree is crossing-free and merges edges smoothly and frequently; it avoids flows through its leaves; its main branches grow as straight and smooth as possible. There is a tradeoff between the smoothness and frequency of merges, but spiral trees allow these properties to be balanced through the choice of restricting angle. We therefore first use our approximation algorithm [3] to compute a spiral tree S on the root and the leaves. The spiral tree S is crossing-free and avoids its leaves by the use of obstacles. However, it is by definition thin and does not have a smooth appearance when avoiding leaves or obstacles. Below we describe how to thicken and smooth S to create the flow tree T (see Fig. 4). During this process we preserve the topology of S, that is, T has the same leaf order and internal structure as S and passes obstacles on the same side. After thickening S we smooth and generally improve the resulting tree T by minimizing the weighted sum $F(T)$ of a carefully chosen set of cost functions that smoothen the tree ($F_S(T)$) and straighten main branches ($F_{str}(T)$), while still avoiding obstacles and leaves ($F_{obs}(T)$) and approximately maintaining the angles ($F_{AR}(T)$ and $F_B(T)$). Our algorithm can be summarized as follows:

($\mathcal{P}$ = points, $\mathcal{W}$ = weights)
Algorithm FlowTree($\mathcal{P}, \mathcal{W}, \alpha$)
1. $S \leftarrow$ ComputeSpiralTree($\mathcal{P}, \alpha$) *(See [3])*
2. $T \leftarrow$ Subdivide and thicken S using $\mathcal{W}$
3. Minimize $F(T)$ *(see Section 4.2 and 4.3)*

The remainder of this section is structured as follows. In Section 4.1 we explain the structure of our trees and discuss the different types of nodes. In Section 4.2 we describe our cost functions and in Section 4.3 we explain how to minimize the resulting global cost function. Finally, in Section 4.4 we discuss the spiral splines which we use to draw the finalized tree.

4.1 Tree structure

Spiral trees and flow trees are both directed trees, with the root as the source. As mentioned above, it is convenient for our computations to conceptually direct these trees from the leaves towards the root. When drawing the final flow trees we of course reverse the direction again. Each node of our trees (except for the root) has exactly one outgoing edge to its *parent*. Similarly, all nodes which are not leaves have incoming edges from their *children*. Our trees have four different types of nodes (see also Fig. 4):

root node: the root, the only node without parent.

leaf node: the leaves, the only nodes without children.

join node: nodes where edges merge. All join nodes have one parent and at least two children.

subdivision node: nodes that subdivide edges, used to avoid obstacles. These nodes have one parent and one child.

The thick flow tree T generally needs more subdivision nodes to avoid obstacles than its thin counterpart S. Hence, before minimizing our cost functions, we further subdivide T until the length of every edge is below a specified threshold (see Section 4.3 for details). Because the edges of T are thick, the children of a join node ν should not connect directly with ν, but with special *dummy nodes* at a slightly offset positions (see the black nodes in Fig. 5). We compute these dummy nodes for every child of a join node. They act as the true endpoints of the edges between a join node and its children.

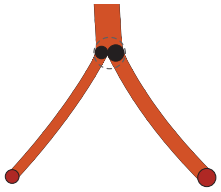

Fig. 5. Dummy nodes.

As stated in Section 3, the edges of spiral trees, and hence also of flow trees, consist of (segments of) β-spirals: logarithmic spirals with angle β. An edge $e = (p, q)$ between a node p and its parent q is the β-spiral segment between p and q, such that the absolute value of β is minimal. This uniquely defines e.

4.2 Cost functions

In this section we describe a set of cost functions, which we believe capture the properties of high-quality flow trees. These cost functions are defined on the nodes and edges of the flow tree T. We combine the separate functions to one global cost function $F(T)$, which we optimize by moving the join and subdivision nodes. The positions of the root and the leaves have to remain fixed.

Obstacle cost. We distinguish two types of obstacles: polygon obstacles and leaf nodes. We use polygon obstacles to model important map features that need to be avoided by the flow tree. We include leaf nodes as obstacles to ensure that the flow tree avoids these nodes and hence all branches are visible. The thin spiral tree S avoids all obstacles. The obstacle cost ensures that also the thick flow tree T does not overlap with the obstacles. It is also often aesthetically more pleasing if the thick edges are clearly separated from the obstacles. Therefore

we add a buffer around every obstacle (see Fig. 6). The choice of a proper buffer size B strongly depends on the map. Although we can easily compute a reasonable value for B, this value can be changed interactively.

We now describe the obstacle cost for a single node p. Consider any obstacle Ω and let q be the point on the obstacle closest to p. For leaf nodes, q is simply the position of the leaf. Let D be the distance between p and q. Further, let t be half the thickness of the flow map at p. If Ω is a leaf node, t also includes the thickness (weight) of this leaf. The obstacle cost for the node p and the obstacle Ω is as follows:

$$F_{obs}(p,\Omega) = \begin{cases} \frac{t}{BD}(\frac{B}{2}+t) + \frac{D}{Bt}(\frac{B}{2}-t), & \text{if } D < t; \\ (1-\frac{D-t}{B})^2, & \text{if } t \le D \le t+B; \\ 0, & \text{otherwise.} \end{cases}$$

The obstacle Ω and the flow tree at p overlap when $D < t$. When there is no overlap, but p still falls within the buffer of Ω, the cost simply increases quadratically as D decreases. When $D = t$, the cost equals one. When $D < t$, the cost increases asymptotically to infinity as D approaches zero. This ensures that the flow tree does not "jump" over obstacles and hence maintains the same topology as the spiral tree S. The formula for $D < t$ ensures that F_{obs} is C^1-continuous.

There is one special case: let Ω be a leaf node and consider the chain of subdivision nodes until the first encountered join node (see Fig. 6). We define the obstacle cost for these subdivision nodes with respect to Ω to be zero. The join node is handled as usual.

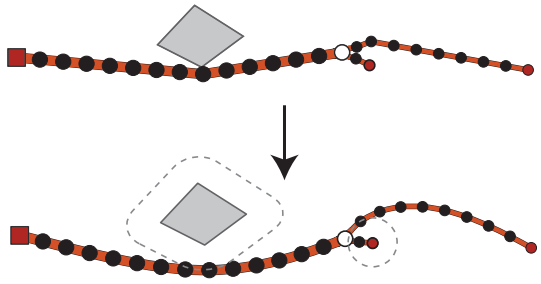

Fig. 6. Optimizing subdivision nodes. Buffers are dashed.

The obstacle cost for the tree T is simply the sum of the costs for all combinations of subdivision and join nodes of T and obstacles.

Smoothing cost. The main purpose of the subdivision nodes is to make the flow tree appear smooth. Hence we define a smoothing cost for subdivision nodes. Let p be a subdivision node and let β_1 and β_2 be the angles of the edges incident on p. Then the smoothing cost for p is defined as:

$$F_S = (\beta_1 - \beta_2)^2$$

Note that if $F_S = 0$ for a certain subdivision node, both incident edges follow the same spiral, which gives a smooth appearance. The smoothing cost for the tree T is again simply the sum of all smoothing costs of all subdivision nodes of T.

Angle restriction cost. For the appearance of the flow tree it is important that the angle restriction, established by the spiral tree S, is maintained at least approximately at join nodes. To that end, we define an angle restriction cost for join nodes. Let p be a join node, and let β_1 and β_2 ($\beta_1 > \beta_2$) be the angles of the edges to the children of p. Then the angle restriction cost is defined as:

$$F_{AR} = \log(\sec\beta_1) + \log(\sec\beta_2)$$

The choice of the function $\sec(\beta)$ is directly related to the properties of spiral trees (see [3]). Since the function $\sec(\beta)$ grows very rapidly, we use the logarithm to be able to combine it more easily with the other cost functions. Nonetheless, the cost grows to infinity as the angles approach $\pi/2$. This ensures that the distance to the root monotonically decreases along the edges of a flow tree.

Balancing cost. The angle restriction cost does not involve the specific angle α that is used to compute the spiral tree. We would like the cost of a flow tree to be minimal when $\beta_1 = \alpha$ and $\beta_2 = -\alpha$ (see Fig. 7). Therefore we add the following balancing cost:

$$F_B = 2\tan^2(\alpha)\log(\csc((\beta_1-\beta_2)/2))$$

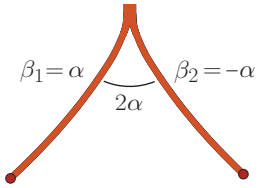

Fig. 7. Optimal join node.

One can prove that, with this definition, the minimal cost is obtained by the tree in Fig. 7.

Straightening cost. The angle restriction and balancing costs cover only the edges from a join node to its children. We need a separate cost for the edge from a join node to its parent. Let β be the angle of this edge. A reasonable value for β would appear to be $(\beta_1 + \beta_2)/2$. However, if a thick edge merges with a thin edge, then it is visually more pleasing if β equals the angle of the thick edge (see Fig. 8). This allows us to easily follow the main branches of the flow tree. Let β^* be the optimal angle for β, then the straightening cost is:

$$F_{str} = (\beta - \beta^*)^2$$

We need to determine a good value for β^*. We describe this computation for an arbitrary number of children. Let $\beta_1, \ldots, \beta_n$ be the angles of the edges to the children and let $t_1, \ldots, t_n$ be the corresponding thicknesses with t^* as maximum. We say that an edge is relevant if $t_i \ge ct^*$. (We use $c = 0.5$ for our maps.) The optimal angle β^* is:

$$\beta^* = (\sum_{i|t_i \ge ct^*} t_i\beta_i)/(\sum_{i|t_i \ge ct^*} t_i)$$

Global cost function. The global cost function is now simply a linear combination of the different costs defined above:

$$F(T) = c_{obs}F_{obs}(T) + c_S F_S(T) + c_{AR}(F_{AR}(T) + F_B(T)) + c_{str}F_{str}(T)$$

The constants are all positive and can be tuned to achieve a suitable tradeoff between the different costs. Specifically, we use $c_{obs} = 2.0$, $c_S = c_{str} = 0.4$, and $c_{AR} = 0.077$. In principle, these parameters could also be set differently for different parts of the tree, thus allowing a user to optimize the tree locally. However, local parameters would be difficult to integrate into an intuitive user interface.

4.3 Minimizing the global cost function

Our goal is now to smoothen and straighten the thick flow tree T by minimizing the global cost function $F(T)$. The value of $F(T)$ is completely determined by the positions of the nodes of T. Since the root and the leaves of T have to remain fixed, we minimize $F(T)$ by moving the subdivision and the join nodes. For the subdivision nodes we fix the distance to the root, that is, subdivision nodes can only rotate around the root. Join nodes are allowed to move freely.

To minimize the global cost function, we use the method of steepest descent. Let T_i be the tree after i iterations (with $T_0 = T$). Consecutive iterations are computed as follows:

$$T_{i+1} = T_i - \varepsilon\nabla F(T_i)$$

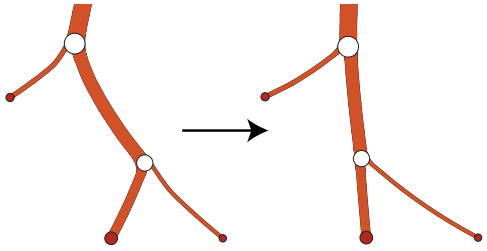

Fig. 8. Straightening join nodes.

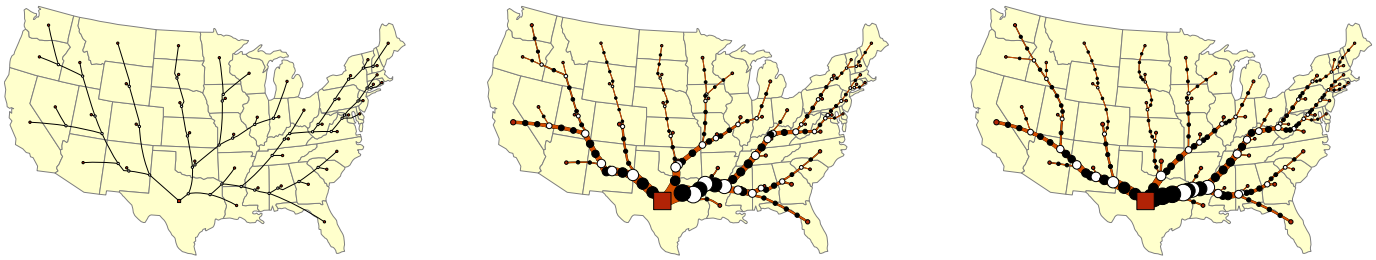

Fig. 9. Flow map illustrating migration from Texas 1995–2000 after computing the spiral tree (left), subdividing and thickening the edges (middle), and finally minimizing the cost function (right).

We ensure that $F(T_{i+1}) < F(T_i)$ and choose ε adaptively. We do need to guarantee that the topology of T_{i+1} is the same as that of $T(i)$. For every edge e of T and every point p, either a leaf node or an obstacle vertex, we need to ensure that e stays on the same side of p. Using a binary search, we can compute the value of ε for which e crosses p. Let the minimum of all these values be ε^*: the smallest step size that results in a different topology. We bound ε by $\varepsilon^*/2$.

The method of steepest descent computes a local minimum, which is sufficient in our case. Since we fix the topology of the flow tree, we expect every local minimum to be of good quality. This is confirmed by our experiments. The method of steepest descent has certain well-known drawbacks, such as zigzagging and slow convergence. In the future we will extend our implementation to use the non-linear conjugate gradient method, which should alleviate some of these issues.

Subdividing and merging. As mentioned in Section 4.1, before minimizing $F(T)$ we add subdivision nodes to T until the length of every edge is below a specified threshold. However, T might not need the same number of subdivision nodes everywhere. So initially we use a fairly sparse subdivision, but we allow adaptive subdivision during optimization. We often need additional subdivision nodes around obstacles. If a point on an edge e of T moves too close to an obstacle, we subdivide e. Subdividing an edge can increase the cost of the resulting tree. However, we also ensure that not too many nodes are added to T, by subdividing an edge only if its length is above a certain threshold. In this way we ensure that the optimization still converges.

Fig. 10. Merging join nodes. Detail of Fig. 2 (top-right).

Next to adding nodes, it is also possible that two nodes merge. A node merges with its parent as soon as the length of the corresponding edge falls below a certain threshold. Two subdivision nodes never merge, as they have a fixed distance to the root. A join node can merge with a subdivision node, in essence removing the subdivision node. It is also possible that two join nodes merge (see Fig. 10), the resulting join node has more than two children. It is straightforward to extend the cost functions to join nodes with an arbitrary number of children. The straightening cost is already defined for an arbitrary number of children and for the angle restriction cost and the balancing cost we simply use the two outermost children.

4.4 Drawing flow maps

Here we explain how to draw the optimized flow tree. We depict the root node by a square of size proportional to the total flow. The leaf nodes are indicated by proportional or graduated circles whose size is proportional to the value of the line ending there. These circles are not essential to our maps and can for instance be replaced by (classed) symbols of different shapes. We do not label our nodes, but use boundary information instead, since cartographers consider this to be more useful for interpreting pattern information [20].

To draw the edges, we introduce a novel type of spline, the *spiral spline*. Recall that the edges are segments of β-spirals. We could use Catmull-Rom splines, but we prefer a method that more accurately draws β-spirals. Spiral splines are in essence cubic Hermite splines. Hermite splines require a tangent at every point. The tangents of points on a spiral should match the tangents of the spiral. Hence, we use logarithmic spirals to compute these tangents for our spiral splines. Consider a subdivision node p and let β_1 and β_2 be the angles of the edges incident on p. The direction of the tangent vector at p is simply the tangent of the β-spiral at p, where $\beta = (\beta_1 + \beta_2)/2$. The length of the tangent vector is $\|p_2\| - \|p_1\|$, where p_2 is the child of p and p_1 is the parent of p. For a target or join node p, the angle β is simply the angle of the edge from p to its parent. Using the spiral splines, the resulting trees appear very smooth.

5 Experimental evaluation

In this section we discuss the results of our algorithm, also in comparison with previous work. We first demonstrate our algorithm by an example and evaluate the influence of the various parameters. All of the results in this section were obtained automatically using our algorithm from Section 4 without user interaction. Only the drop shadows were manually added in a post-processing step. In Section 6 we show further results with additional user-specified input. In Fig. 9 we give an overview of our algorithm using, as an example, migration from Texas. The spiral tree has a nice structure but is not smooth at internal nodes. Thickening shows another issue, namely that main branches deviate around the leaf nodes. Both issues are resolved in the final flow tree, while keeping the structure of the spiral tree. In Fig. 11 we show results of our algorithm using different parameters. From left to right we varied our most important parameter, the restricting angle α. We use $\alpha = 15°$ (left), $\alpha = 25°$ (middle), and $\alpha = 35°$ (right). Lower values for α result in "straighter" flow trees, but can also cause some weaving. Higher values for α often result in a better tree structure, but these flow trees require sharper turns. The top figures use, besides α, our standard parameters. In the bottom figures we show the effects of some of the parameters. In the left figure we increased the buffer size B by a factor of 1.5. This makes the nodes better visible, but does require longer detours. In the middle figure we set $c_S = 0$. Although the flow tree is still drawn smoothly due to the splines, it does not appear smooth anymore. In the right figure we set $c_{str} = 0$. We think the straightening results in a visually more pleasing flow tree. The remaining parameters c_{obs} and c_{AR} must be larger than zero for the algorithm to work properly but otherwise their values seem to have little influence on the result.

The weights of the leaves also influence the layout of the flow map. The following two maps in Fig. 12 show the same data set, namely migration from Norway, at two different points in time. Both maps are based on the same spiral tree, that is, they have the same tree topology. This makes it easy to compare the flows. Differences between the maps cannot only be detected by comparing the widths of flow lines, but are also emphasized by the straightening of main branches. This for instance emphasizes the large flow to Poland in 2009.

Next, we visually compare our results to those of Phan *et al.* [16]. In Fig. 13 the map by Phan *et al.* contains many crossings, the grouping of nodes is somewhat unnatural, and the edges are often routed in a counter-intuitive manner (see also the maps in Fig. 2, Fig. 14, and Fig. 15). Consider for example North Dakota, which is not grouped

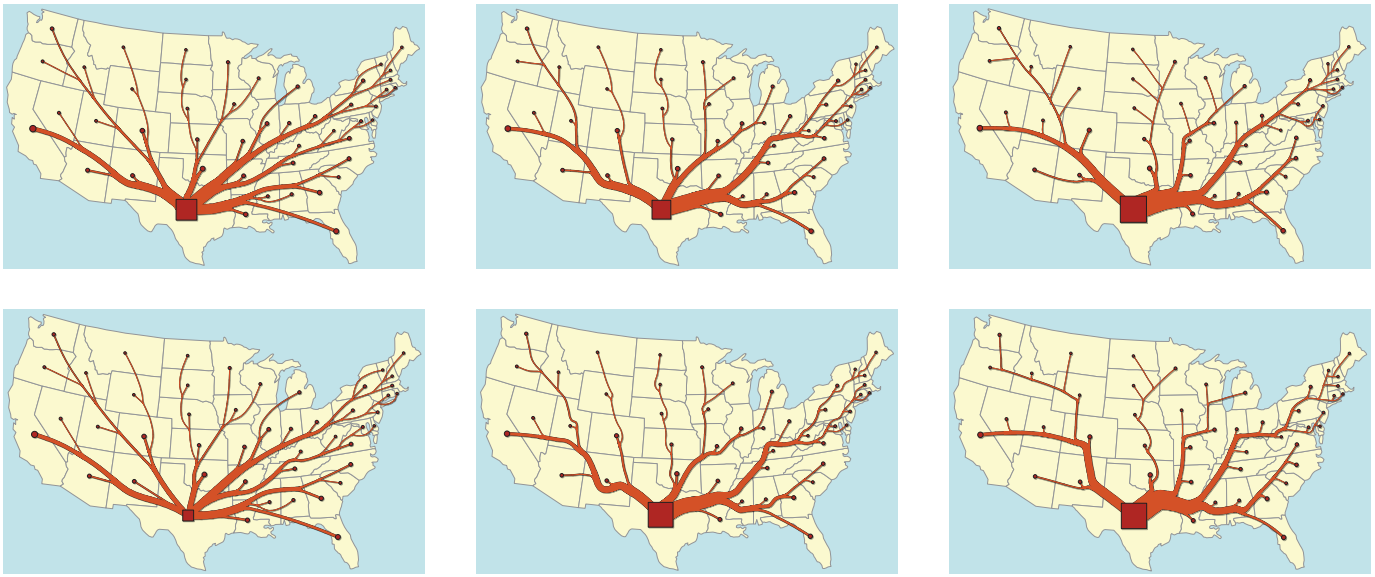

Fig. 11. Various results of our algorithm. We use $\alpha = 15°$ (left), $\alpha = 25°$ (middle), and $\alpha = 35°$ (right). The top figures use standard parameters. For the bottom figures, we increased buffer size B by a factor of 1.5 (left), set $c_S = 0$ (middle), and set $c_{str} = 0$ (right).

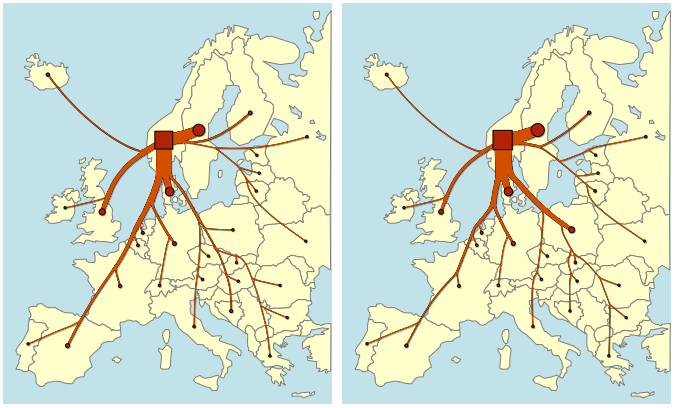

Fig. 12. Migration from Norway in 1996–2000 (left) and 2009 (right).

with South Dakota but instead crosses the branch to South Dakota. Another example is North Carolina, which is reached with a branch that grows from the south via Florida and below Texas. In contrast, our map is crossing-free, smoothly bundles lines, and the main branches of the tree are easy to follow. Our algorithm automatically creates nodes of higher degree which helps to de-clutter the busy East coast.

6 Extensions

We next present various extensions to our automatic method, in particular, user-specified clusters, routing around obstacles, the overlay of flow trees, and waypoints. While in Fig. 13 the naturally looking clustering is simply induced by the spiral tree, we can also accommodate user-specified clusters.

Fig. 14 and Fig. 15 show CO_2 emissions. Fig. 15 was produced by Moran and his co-workers [14, 19] using the algorithm by Phan *et al.* The original data is not available anymore, so our corresponding map in Fig. 14 is based on Moran's most recent data.[1] In Fig. 15 the many crossings in Europe are very noticeable. The cluttering in that map is to some extent due to a large flow from Asia that was routed through Europe. This can be avoided by appropriate clustering. In Fig. 14 we clustered the leaves according to the United Nations geoscheme. This scheme clusters by region (continent). To emphasize the two largest individual flows (US and China) we clustered these on subregion level, i.e., as Northern America and Eastern Asia. In Europe we used subregions as clusters, but placed Ireland, Iceland, Spain, and Portugal outside of the clusters to allow other flows to pass. In comparison it is noticeable that the map in Fig. 15 shows nearly no variation in the width of the flow lines, although the data indicates huge flows from the US. This might be an artifact of logarithmic scaling or the upper bounds on flow width imposed by Phan *et al.* In contrast, our algorithm is able to accommodate linearly scaled flows, even for such a

[1] D. D. Moran, personal communication, 2011.

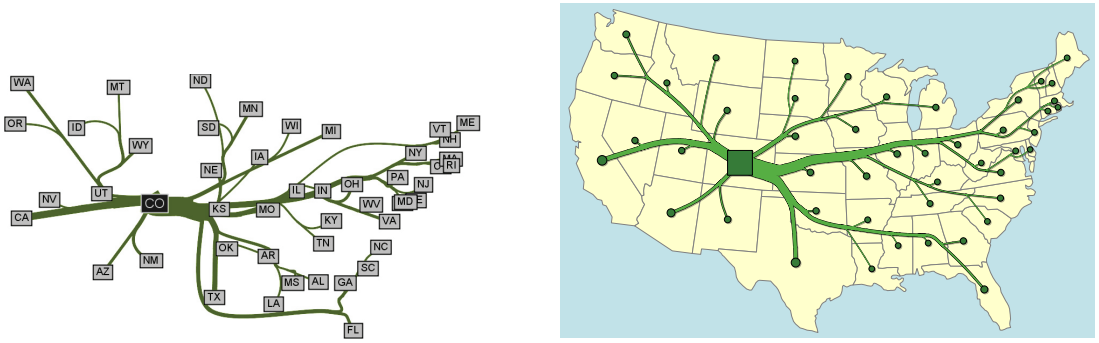

Fig. 13. Two flow maps illustrating migration from Colorado 1995–2000. Phan *et al.* [16] (left), and the output of our algorithm (right).

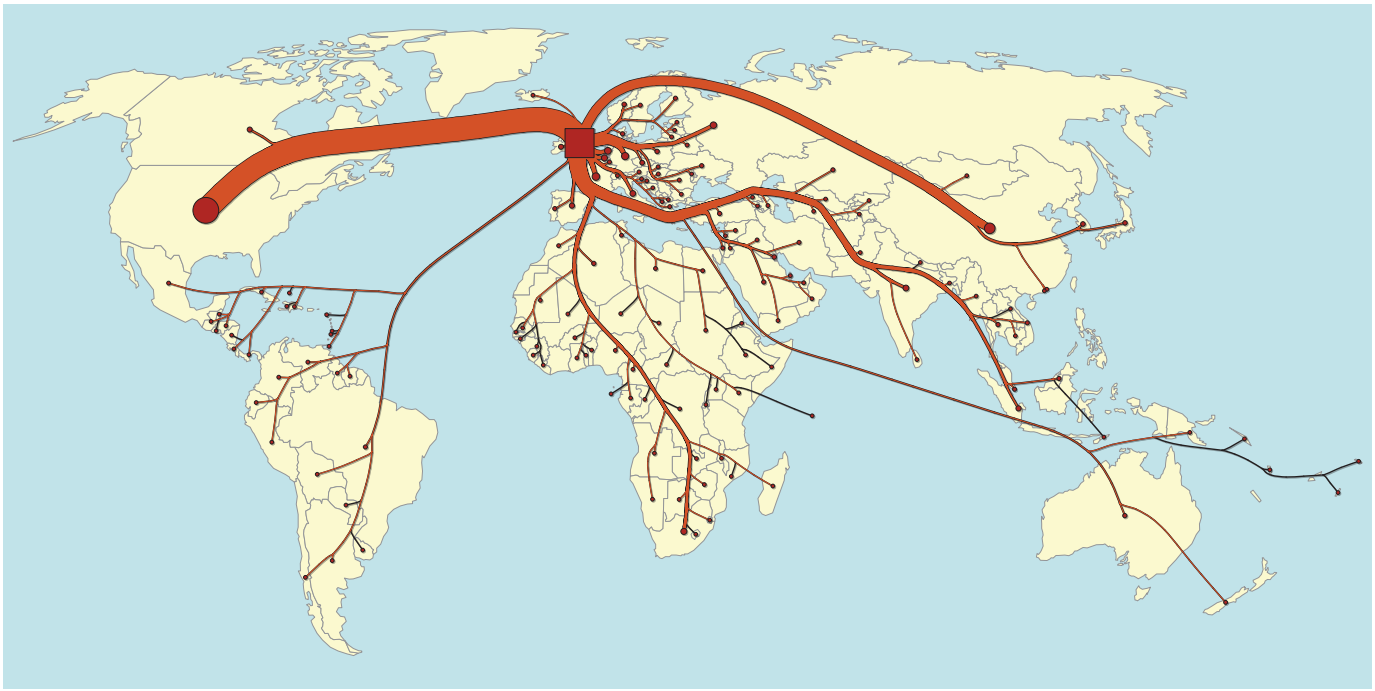

Fig. 14. Flow of embodied CO_2 to the United Kingdom. Embodied CO_2 refers to the entire amount of CO_2 emitted for the production and transportation of goods to consumers. The map shows the CO_2 flows into the United Kingdom from each of its trading partners. CO_2 emissions from transportation are shown as originating from the country that provides the transportation fuel. The embodied CO_2 flows in this map have been calculated using the Eora MRIO Model being built at the University of Sydney.

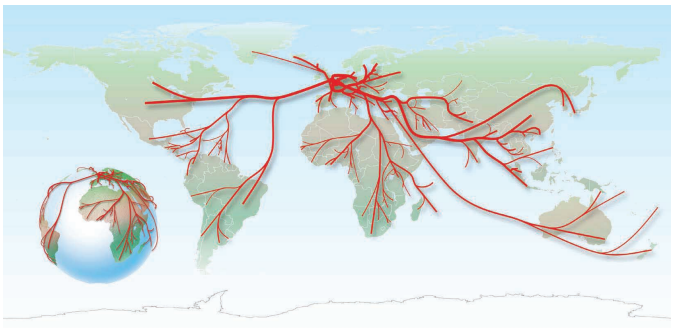

Fig. 15. Total ecological footprint of imports to the UK [19] created using the algorithm by Phan *et al.* [16].

large map where values differ substantially. Zooming in on Europe it is apparent that our maps work well on different scales. The merging of flows in general bears the risk of implying a clustering that might not have a meaning as such. User-specified clusters have the significant advantage of introducing meaningful merges.

Note that Fig. 15 does not route along shipping routes and does not try to avoid crossing landmasses. Our algorithm is able to do so, as illustrated in Fig. 16 and Fig. 19. The map in Fig. 16 (top) is computed without avoiding landmasses, in Fig. 16 (bottom) we added (parts of) the island boundaries as obstacles to the input. Both figures show clearly that the major part of exports from Sulsel is directed towards Sumatra and Java. However, the flows in the top map cover nearly the complete eastern coastline of Sumatra and cross various smaller islands, whereas the lower map gently curves through the Strait of Malacca and around the islands. The difference between the two maps is somewhat subtle, since both were created with the same algorithm, but the avoidance of landmasses increases the legibility of the map.

Flow maps can also contain more than one flow tree. In Fig. 17 we again compare our map against a map produced by Phan *et al.*

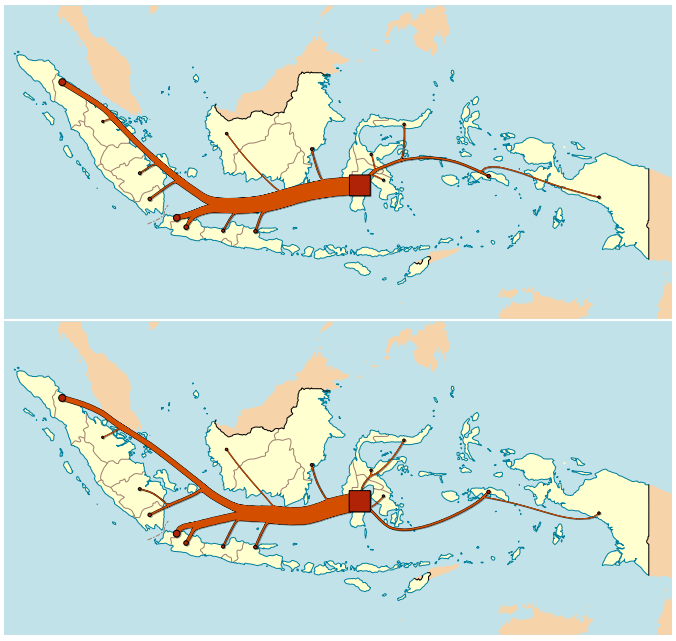

Fig. 16. Exports from the province Sulsel on Sulawesi to all other provinces of Indonesia in 1990. Without avoiding landmasses (top) and avoiding landmasses (bottom).

Their paper uses data from the U.S. census bureau, which would imply that this map depicts the top 10 states that migrate to California and New York 1995–2000. Unfortunately, the states that they show, do not correspond to this data. Our map shows the correct data set. Both maps were created by overlaying two single source flow trees. Our current system does not support the simultaneous creation of more than one flow tree, but we simulated the process by adding parts of the

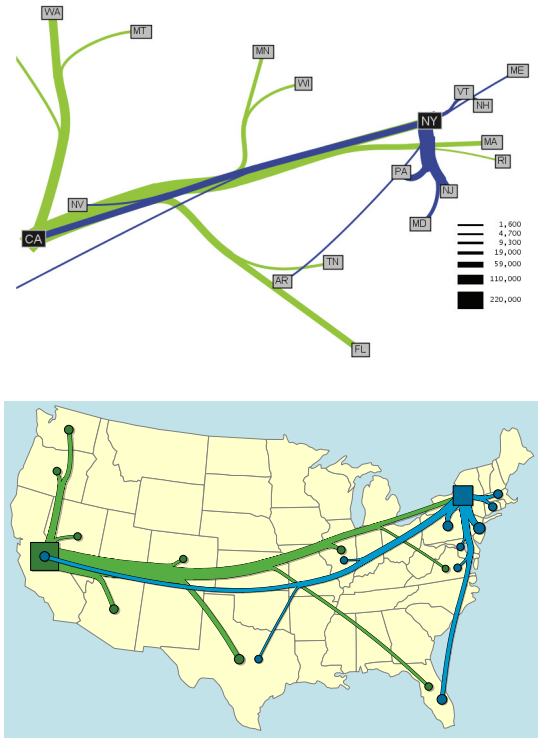

Fig. 17. Two flow maps illustrating the top 10 states that migrate to California and New York 1995–2000. Phan *et al.* [16] (top), and the output of our algorithm (bottom).

first tree (green) as obstacles for the input of the second (blue). The result is already satisfactory, however, a branching point of the blue tree coincides with a branch of the green tree. The green tree, being the first to be computed, did not know about this branching point and hence could not avoid it. How to integrate the creation of several flow trees is a challenging open question. Another issue is that our flow maps do not indicate the direction of the flow. Additional experiments

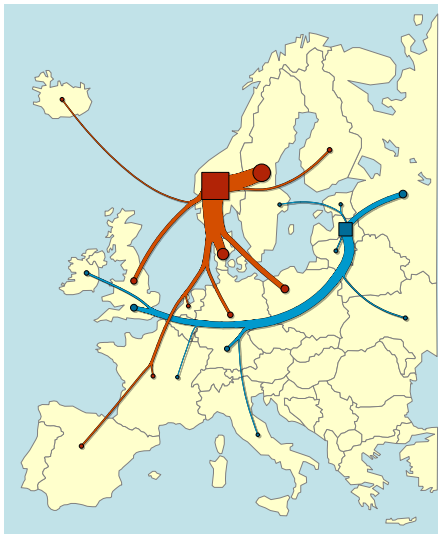

Fig. 18. Migration from Norway and Latvia in 2009.

with symbolism, for example, arrows at leaves or along major flow lines, or using squares for destinations and circles for origins, might lead to an effective method to indicate flow direction.

The data set depicted in Fig. 17 is particularly challenging and somewhat atypical for flow maps that contain several flow trees. The main axis California–New York is contained in both trees and bound to conflict. In Fig. 18 we show the overlay of two flow trees for a more typical data set, namely migration from Norway and from Latvia. Here we used the same approach, namely using parts of the red tree as obstacle for the blue tree, to much greater success. The map clearly shows the importance of neighboring countries for migration, but also highlights the UK as a popular destination. One issue remains though. To be able to use parts of the first flow tree as obstacles for the second flow tree, we manually moved the leaf locations for regions that occur in both flow trees (France, Germany, UK, Sweden). This can be confusing, especially if the two locations are separated by a flow line.

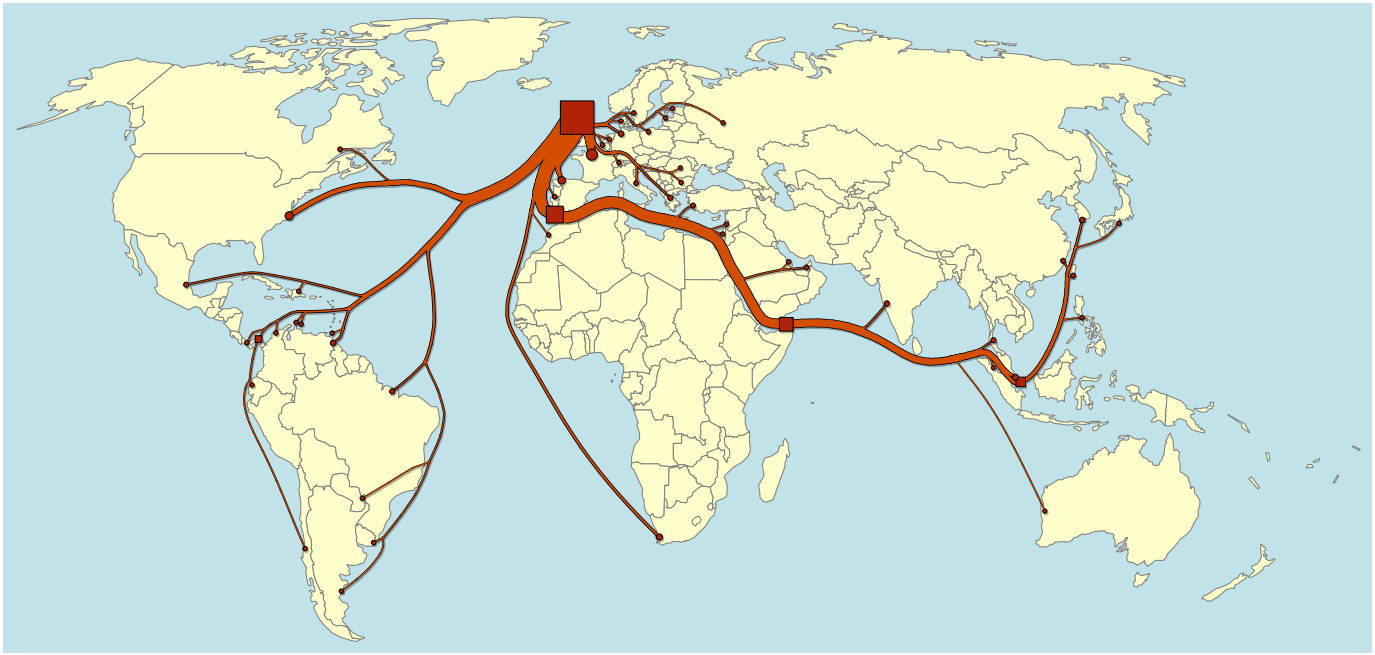

Fig. 19. Top 50 whisky exports from Scotland in 2009 by volume.

Our final map in Fig. 19 shows the top 50 countries to which Scotland exported whisky in 2009. Much of the whiskey exported in 2009 from Scotland was transported by deep-sea container ships, much of the whiskey exported to Europe (with France as main importer) was transported across the Channel at Dover. We therefore route flows in this map along shipping routes, by using landmasses as obstacles. Shipping routes typically use shortest paths, which integrates well with the optimization criteria for our flow trees. However, shipping routes might need to deviate in direction substantially from the direct connection between a source and a destination, for example, if they have to pass through an important passage. In these situations we cannot directly enforce the angle restriction for all nodes. We handle this issue by introducing *waypoints*: intermediate destinations, which themselves serve as the root of a subsidiary flow tree (waypoints are indicated as squares in Fig. 19). For shipping routes we have a canonical set of potential waypoints corresponding to strategic maritime passages [18]. For the map in Fig. 19 we used four waypoints corresponding to the Panama Canal, the Strait of Malacca, the Strait of Bab el-Mandab and Gibraltar. The aggregation effect of the flow map shows the size of the Latin American and the Caribbean market, a fact that would have been difficult to deduce from the raw data. Furthermore, it emphasizes the importance of the Suez Canal for European trade. Note that the waypoints are shown only to illustrate the composition of the tree and would not be part of the final map. Flow maps by design bear the risk of misinterpretation. This risk is particularly high when flows approximately follow trade routes. We have no exact data for the trade routes but even with this data a flow tree would deviate from it. Our map, for instance, is not meant to indicate that whisky to Italy is transported via Germany.

7 Implementation

We created a proof-of-concept implementation of our flow map algorithm in Java. Our program can compute a spiral tree from a given point set, also including obstacles. Next to that, our program contains features that allow interactive editing of spiral trees. We have also implemented the cost function minimization, as described in Section 4. Finally, our program supports export of flow trees in vector format. Our current implementation does not support multiple flow trees or waypoints (see Section 5). Such flow maps were created by manually overlaying single flow trees created by the program. Figures in this paper with single flow trees were fully automatically created using our program, except for some minor editing (drop shadows, clipping, etc.). Our program computes an optimal flow tree for most maps in less than a minute on a system with a Pentium D 3 GHz processor (dual-core) and 1 GB of RAM. World maps require a couple of minutes.

Our data stems from various sources which we summarize in the following: Tobler [1], Moran (personal communication), Stelder [21], the Indonesian central planning bureau BAPPENAS (`www.regroningen.nl/irios_tables.shtml`), Statistics Norway (`www.ssb.no`), Central Statistical Bureau of Latvia (`www.csb.gov.lv`), and the Scotch Whisky Association in Edinburgh, statistical report 2009 (`www.scotch-whisky.org.uk`). All URLs were accessed on 2011-04-19.

8 Conclusion

In this paper we introduced an algorithm that produces flow trees of high visual quality. Our flow trees are based on spiral trees, a novel type of Steiner tree. As a consequence, our flow trees are crossing-free, merge smoothly, and naturally cluster. The main branches of our flow trees grow as straight and smooth as possible, for an overall organic impression that makes it easy to interpret the maps. Furthermore, our trees allow for high-degree internal nodes, can avoid obstacles as well as their own leaves, and do not need to distort the base map.

An interesting and challenging direction for future work is the integrated creation of several flow trees while minimizing crossings between branches of different trees. In those cases where crossings are unavoidable, we prefer crossings among lesser branches of the trees.

Acknowledgments

We thank Dan Moran for providing us with the updated data on embodied CO_2 flow, we thank Doantam Phan for discussing his results with us, and we thank Jack van Wijk for helpful discussions on the topic of this paper. Bettina Speckmann and Kevin Verbeek are supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.022.707.

References

[1] CSISS - Spatial Tools: Tobler's Flow Mapper. `http://www.csiss.org/clearinghouse/FlowMapper/`.

[2] I. Boyandin, E. Bertini, and D. Lalanne. Using flow maps to explore migrations over time. In *Proc. Workshop in Geospatial Visual Analytics: Focus on Time (GeoVA(t))*, 2010.

[3] K. Buchin, B. Speckmann, and K. Verbeek. Angle-restricted Steiner arborescences for flow map layout. In *Abstr. 27th European Workshop on Computational Geometry*, pages 163–166, 2011.

[4] K. C. Cox, S. G. Eick, and T. He. 3D geographic network displays. *SIGMOD Rec.*, 25:50–54, December 1996.

[5] W. Cui, H. Zhou, H. Qu, P. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics (INFOVIS 08)*, 14(6):1277–1284, 2008.

[6] B. D. Dent. *Cartography: Thematic Map Design*. McGraw-Hill, New York, 5th edition, 1999.

[7] C. A. Duncan, A. Efrat, S. G. Kobourov, and C. Wenk. Drawing with fat edges. *International Journal of Foundations of Computer Science*, 17(5):1143–1163, 2006.

[8] A. M. Francis and J. B. Schneider. Using computer graphics to map origin-destination data describing health care delivery systems. *Social Science & Medicine*, 18(5):405–420, 1984.

[9] D. Guo. Flow mapping and multivariate visualization of large spatial interaction data. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1041–1048, 2009.

[10] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics (INFOVIS 06)*, 12(5):741–748, 2006.

[11] D. Holten and J. J. van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, 2009.

[12] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*, volume 53 of *Annals of Discrete Mathematics*. Elsevier, 1992.

[13] J. Krozel, C. Lee, and J. Mitchell. Turn-constrained route planning for avoiding hazardous weather. *Air Traffic Control Quarterly*, 14(2):159–182, 2006.

[14] D. D. Moran, M. C. Wackernagel, J. A. Kitzes, B. W. Heumann, D. Phan, and S. H. Goldfinger. Trading spaces: Calculating embodied ecological footprints in international trade using a product land use matrix (PLUM). *Ecological Economics*, 68(7):1938–1951, 2009.

[15] T. Munzner, E. Hoffman, K. Claffy, and B. Fenner. Visualizing the global topology of the MBone. In *Proc. IEEE Symposium on Information Visualization (INFOVIS '96)*, pages 85–92, 1996.

[16] D. Phan, L. Xiao, R. Yeh, P. Hanrahan, and T. Winograd. Flow map layout. In *Proc. IEEE Symposium on Information Visualization (INFOVIS '05)*, pages 219–224, 2005.

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

[18] J.-P. Rodrigue, C. Comtois, and B. Slack. *The Geography of Transport Systems*. Routledge, New York, 2nd edition, 2009.

[19] A. Simms, D. D. Moran, and P. Chowla. *UK Interdependence Report*. New Economics Foundation, London, 2006.

[20] T. A. Slocum, R. B. McMaster, F. C. Kessler, and H. H. Howard. *Thematic Cartography and Geovisualization*. Pearson, New Jersey, 3rd edition, 2010.

[21] D. Stelder. Interregional I-O modelling in Indonesia: a constrained growth rate approach with endogenous investment. *Economic Systems Research*, 7(2):117–135, 1995.

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

[23] J. Wood, J. Dykes, and A. Slingsby. Visualization of origins, destinations and flows with OD maps. *The Cartographic Journal*, 47(2):117–129, 2010.