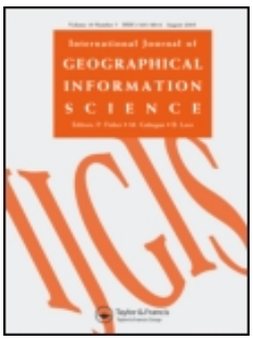


International Journal of Geographical Information Science

ISSN: 1365-8816 (Print) 1362-3087 (Online) Journal homepage: http://www.tandfonline.com/loi/tgis20

Interactive focus maps using least-squares optimization

Thomas C. van Dijk & Jan-Henrik Haunert

To cite this article: Thomas C. van Dijk & Jan-Henrik Haunert (2014) Interactive focus maps using least-squares optimization, International Journal of Geographical Information Science, 28:10, 2052-2075, DOI: 10.1080/13658816.2014.887718

To link to this article: http://dx.doi.org/10.1080/13658816.2014.887718

International Journal of Geographical Information Science, 2014
Vol. 28, No. 10, 2052–2075, http://dx.doi.org/10.1080/13658816.2014.887718

Taylor & Francis
Taylor & Francis Group

Interactive focus maps using least-squares optimization

Thomas C. van Dijk* and Jan-Henrik Haunert

Lehrstuhl für Informatik I, University of Würzburg, Würzburg, Germany

(*Received 3 July 2013; accepted 17 January 2014*)

We present a new algorithm that enlarges a focus region in a given network map without removing non-focus (i.e., context) network parts from the map or changing the map's size. In cartography, this problem is usually tackled with fish-eye projections, which, however, introduce severe distortion. Our new algorithm minimizes distortion and, with respect to this objective, produces results of similar quality compared to an existing algorithm. In contrast to the existing algorithm, the new algorithm achieves real-time performance that allows its application in interactive systems. We target applications where a user sets a focus by brushing parts of the network or the focus region is defined as the neighborhood of a moving user.

A crucial feature of the algorithm is its capability of avoiding unwanted edge crossings. Basically, we first solve a least-squares optimization problem without constraints for avoiding edge crossings. The solution we find is then used to identify a small set of constraints needed for a crossing-free solution and, beyond this, allows us to start an animation enlarging the focus region before the final, crossing-free solution is found. Moreover, memorizing the non-crossing constraints from an initial run of the algorithm allows us to achieve a better runtime on any further run – assuming that the focus region does not move too much between two consecutive runs. As we show with experiments on real-world data, this enables response times well below 1 second.

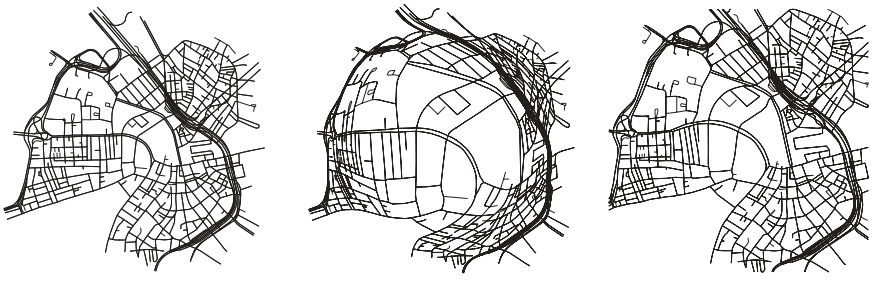
Keywords: interactive cartography; focus-and-context maps; least-squares optimization; algorithms

1. Introduction

Users of navigation maps usually focus on certain map regions, for example, the neighborhood of their current location. In order to help a user read the important information of a map quickly, we aim to enlarge the focus regions in a given map by a certain factor. At the same time, however, we want to display the non-focus regions of the given map – this is important to provide the user with context – and we do not allow the map to be enlarged. Obviously, we cannot reach our goal without distorting the given map. If the user does not need to measure exact distances and the distortions are not too large, however, distortions can be tolerated. In fact, it is very common to visualize metro networks with distorted geometry (Jenny 2006, Nollenburg and Wolff 2011, Wang and Chi 2011), but also distorted maps of road networks have been proposed, for example, to set a focus on crucial decision points of a route (Agrawala and Stolte 2001, Schmid 2008).

In order to display a focus region at a large scale, integrated with context information at a smaller scale, the use of fish-eye projections has been proposed (Harrie *et al*. 2002, Li 2009, Yamamoto *et al*. 2009, Haunert and Sering 2011). An alternative method

*Corresponding author. Email: thomas.van.dijk@uni-wuerzburg.de



(a) Input map of `Boston`. (b) Fish-eye projection. (c) Optimal distortion.

Figure 1. Comparison of focus maps. Figure (a) is the input network. Figure (b) uses a fish-eye projection (Yamamoto *et al*. 2009), and Figure (c) is based on optimization (Haunert and Sering 2011). Focus is on the center of the map.

proposed by Haunert and Sering (2011) is based on optimization – it minimizes distortion – and is both exact and efficient, in the sense that its runtime is polynomial in the input size. See Figure 1 for a comparison on an example map. In practice, this optimization method solves problem instances of typical size in a few seconds. For example, it processed a network of 2821 nodes and 3332 edges representing the city center of Boston in 4.33 seconds. We consider this fast enough for static maps but not for application in interactive navigation systems. In particular, we would like to allow a user to interactively choose a focus region by moving a mouse cursor over a map, and we would like to keep the focus on the position of a moving user. In both the cases, we would like to update the displayed map in real time.

Our new algorithm does not run in real time in the sense that it instantaneously yields the final output map. Instead, as soon as a user selects a focus region, we show an animation that continuously warps the current map into a map in which the focus region is enlarged. This animation takes about 2 seconds and helps the user see where distortion is introduced and to recognize the correspondences between the features of the input and the output map. A nice property of our new algorithm is that we do not need to wait with the start of the animation until the final output map is computed. Thus we win time and create a more interactive impression: the user does not notice that the algorithm is still computing. Moreover, our algorithm updates the output map dynamically, that is, if the center of the focus region has not moved much since the previous run of our algorithm, we do not need to compute the output map from scratch. Instead, we can reuse the information stored with the previous run and, thereby, accelerate the update of the output map.

1.1. Our contribution

In this article we provide a new algorithm for drawing focus maps based on distortion minimization. The algorithm of Haunert and Sering (2011) is based on convex quadratic programming. Our iterative algorithm is based on the least-squares solution of several systems of linear equalities. This is a less powerful subroutine to base an algorithm on, which has the disadvantage that less complexity is hidden in a black box. In return, however, the subroutine is much faster than the old approach. This improved performance allows for qualitatively different applications, since this new algorithm is fast enough for use in interactive systems.

We review related work in Section 2. In Sections 3–5 we develop the basic algorithm. In Section 6 we provide some improvements for use in interactive systems. In Section 7 we provide a detailed evaluation of the performance of the algorithm. It shows that the algorithm is well suited for use in interactive systems. A demonstration video using our implementation of the algorithm is included as supplemental material.

2. Related work

Fish-eye projections have a long history in cartography. More recently, the widespread availability of mobile devices with small screens has renewed interest in them (Harrie *et al*. 2002, Li 2009, Yamamoto *et al*. 2009). A problem with fish-eye projections is the severe distortion they introduce in the area between the enlarged focus and the surrounding context. To remedy this, Haunert and Sering (2011) have formalized a version of focus maps as an optimization problem: enlarge some focus area of the map, while staying within the original frame and minimizing distortion. The algorithm we present in this article is based on this approach, so we review it in more detail later on.

Viewed at a high level, our algorithm is an optimization-based graph-layout method. In this way it is related to other approaches based on network layout. Examples of those include automatic generation of route sketches (Agrawala and Stolte 2001, Schmid 2008), destination maps (Kopf *et al*. 2010), and travel-time maps (Shimizu and Inoue 2003), and the rescaling of dense areas in maps of varying network density (Merrick and Gudmundsson 2006).

The optimization problem that we consider in this article has an objective function that takes the form of least-squares optimization. In this way it is similar to approaches that have been employed for 3D volume visualization (Wang *et al*. 2008) and focus-and-context metro maps (Wang and Chi 2011). Specifically in cartography, least-squares optimization is also used for displacement, that is, increasing the distance between certain map objects (Harrie and Sarjakoski 2002, Sester 2005) and for continuous area cartograms (House and Kocmoud 1998, Inoue and Shimizu 2006). In these works, constraints like those that forbid unwanted edge crossings are typically relaxed (for example in the method of Harrie and Sarjakoski 2002). Haunert and Sering (2011), by contrast, optimize a least-squares objective subject to hard constraints about edge crossings. Our modeling does not have hard constraints, but our algorithm takes special care to guarantee non-crossing; indeed, our algorithmic contribution is mainly how to handle edge crossings in the absence of hard constraints. This general approach of handling crossing constraints explicitly at an algorithmic level, rather than by modeling and the use of a black-box solver, is also seen in graph drawing research (for example, Simonetto *et al*. 2011).

Schmid *et al.* (2010) emphasize the importance of selecting a good set of map objects to provide the user with context, for example, salient landmarks. In the case of road networks, this amounts to road segment selection (Jiang and Claramunt 2004, Schmid *et al.* 2010, Van Dijk and Haunert 2013). In this article, however, we assume that a selection of map objects has been made prior to the execution of our algorithms.

There is some work on the combination of focus maps and feature selection, for example by Cecconi and Galanda (2002) and Hampe *et al*. (2004). The Focus + Glue + Context maps by Yamamoto *et al*. (2009) provide this combination in an interactive system in which the user interacts with a movable focus area. The system distinguishes three areas: focus, glue, and context. There is no distortion within the focus and context areas, but there is severe distortion in the glue area. This can make it hard to read from the map how the roads in the focus area connect to the context area. The system

also involves showing only a selection of the roads in the glue area. This helps reduce visual clutter by reducing the information content of the distorted part of the map but comes at the cost of not displaying the entire network topology.

Roth (2013) reviews the state of science in interactive cartography. In particular, he reports that the recommended response time for cartographic interactions is about 1 or 2 seconds (Haunold and Kuhn 1994, Wardlaw 2010). More generally in human–computer interaction, response times of up to about a second are considered sufficient for real-time interaction without interrupting the user's thinking process (Miller 1968, Card *et al.* 1980). Here we note that, on maps that we consider reasonable for our application, the focus-map algorithm of Haunert and Sering falls outside this margin, whereas we will see in Section 7 that our algorithm is fast enough.

2.1. *Optimal focus and context maps*

Here we review the measure of road network distortion introduced by Haunert and Sering (2011), since we will also use this measure of distortion. Broadly speaking, this measure allows translation and uniform scaling 'for free,' but any other transformation is distortion and, as such, associated with a certain cost. This is defined locally on the nodes of the network, as follows, and the distortion of the network is the sum of the distortion of the individual nodes.

Consider an *input network* $G = (V, E)$ where each node $u \in V$ has a position (X_u, Y_u) and every edge $\{u, v\} \in E$ corresponds to a road segment. This network might itself be a cutout of a larger map, but has to contain all the context that the focus map should provide: for the purpose of any algorithms, this network is all that exists. Let $Dist(u, v)$ be the Euclidean distance between node u and node v.

For every node we have an output position (x_u, y_u) and a local scale s_u. If the neighborhood of a node u undergoes only translation and uniform scaling by factor s_u, then for every neighbor v we have the exact equality

$$s_u \begin{pmatrix} X_v - X_u \\ Y_v - Y_u \end{pmatrix} = \begin{pmatrix} x_v - x_u \\ y_v - y_u \end{pmatrix} \tag{1}$$

Any violation of Equation (1) is defined to be distortion, which is quantified as follows. Let the *discrepancy* of an edge $\{u, v\}$ be

$$\delta_{uv} = s_u \begin{pmatrix} X_v - X_u \\ Y_v - Y_u \end{pmatrix} = \begin{pmatrix} x_v - x_u \\ y_v - y_u \end{pmatrix} \tag{2}$$

Then the distortion of an edge $\{u,v\}$ is defined as $\|\delta_{uv}\|^2$. The distortion of a node is the sum over its adjacent edges, where each edge is weighed inversely by its length. By summing over $N(u)$, the neighbors of u, we get:

$$\text{distortion}(u) = \sum_{v \in N(u)} w(u, v) \cdot \|\delta_{uv}\|^2 \tag{3}$$

where $w(u,v) = 1/Dist(u,v)$.

By fixing the value of s_u for some nodes, the solution can be encouraged to enlarge certain neighborhoods in the network. In fact, without further constraints the minimum-

distortion drawing will then uniformly enlarge the entire network. This is not useful, since we want to show the entire network to the user. Therefore a constraint is added to the model: the output drawing should be contained within the original bounding rectangle. The combination of this objective with this constraint is what makes the optimization problem useful. The focus maps of Haunert and Sering minimize the distortion of the network (to near-optimality) while satisfying this bounding-box constraint and without introducing edge crossings. Unfortunately, their algorithm is too slow for use in real-time interactive systems.

2.2. Least-squares solution of systems of linear equations

Here we briefly review the concept of a *least-squares solution*. Consider a (possibly over-constrained) system of linear equalities. It can be represented as a matrix equation of the form $Ax = b$. The *least squares* solution to this equation is the vector x^* that minimizes $\|Ax * -b\|^2$. Let d_i be the amount by which constraint i is violated, that is, $d = Ax^* - b$. Then, by definition, x^* minimizes $\Sigma_i d_i^2$. For this reason, x^* is called the *least squares* solution.

To emphasize that any of the original equalities may be violated, they are called *soft constraints*. Note that we can weigh the relative importance of constraints by multiplying both the sides of an equation by a constant: equality is unaffected, but d_i changes by the same factor. In the basic framework of least-squares solving, there is no direct support for *hard* constraints, that is, forcing $d_i = 0$ for some i. This can be handled, for example, by using only one actual variable for multiple linearly-related 'conceptual' variables.

It is well known that given A and b, x^* can be found by solving $A^{\mathrm{T}}Ax^* = A^{\mathrm{T}}b$. This equation is not over-constrained and can be solved using standard methods. There exist particularly efficient methods if A and $A^{\mathrm{T}}A$ are sparse (see for example Kraus 2004).

3. Modeling

We now lay the groundwork for the rest of the article by introducing notation and stating our optimization problem. By convention we will use uppercase for constants and lowercase for variables.

3.1. Input and variables

The road network to be distorted is given as a connected graph $G = (V, E)$, which is a constant in terms of the optimization problem. This graph contains both the focus and the context parts of the network: the algorithm will distort this network to create a focus map. In an interactive system where the user can pan across a large map, this graph G contains the part of the network that is to be shown on the screen. Such panning and zooming can be handled, but for the rest of the article we consider G to be fixed.

A *drawing* $\mathcal{D}$ of G is an assignment of positions to the nodes in V. Central to our approach is that our algorithm considers three drawings of the same road network: original, current, and output. An example of three such drawings is given in Figure 2.

- The optimized *output* drawing, where node u has position (x_u, y_u). This drawing is to be determined. See Figure 2c.
- The input *original* drawing, with the actual geographic positions. Each node u has a given position (X_u, Y_u). In an interactive system where the user interacts with the map and the focus changes, still this drawing always remains the same: it is the

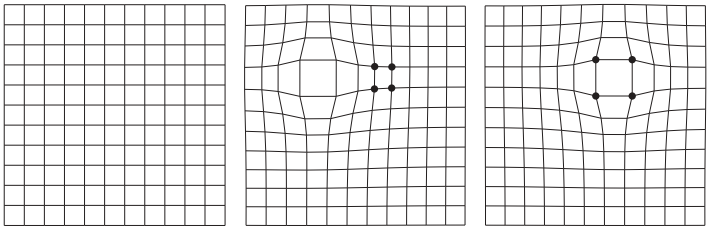


Figure 2. Examples of the three drawings considered by the algorithm. (a) The original input map, a grid in this example. It always stays the same. (b) The current drawing presented to the user, as calculated by the algorithm in a previous run, before the algorithm is run on a new set of focus nodes. Here we indicate the new focus using dots. (c) The output drawing when the algorithm is finished.

geographical ground truth. This drawing must have no edge crossings; in case of bridges or tunnels, the intersections should be added to the graph as nodes. See Figure 2a.

- Finally, also as input, our algorithm considers the *current* drawing, where node u has a given position $\left(X_u', Y_u'\right)$. This means that in an interactive system where the focus changes, a new output drawing is allowed to depend on the drawing currently presented to the user. This allows us to prevent sudden jumps that might disorient a user. Our algorithm also uses this to present an animation from the current to the new output drawing: we can already show some progress before the algorithm has fully completed. See Figure 2b.

An application uses these drawings as follows. When the focus changes, our algorithm calculates a new output drawing. The application can show an animation from the *current* drawing to the newly determined *output* drawing. (In Section 6.1 we show that this animation can be started before the algorithm is fully done.) After the animation to the new drawing has been completed, this output drawing becomes the current drawing for any further computations.

Note that X_u, Y_u, X_u', and Y_u' are constants in our optimization problem. On the other hand, x_u and y_u are variables to be optimized. To determine the distortion of the network, we have a further variable per node: the local scale factor s_u.

At several points in this article, we consider a linear interpolation of the node positions of the *current* drawing to another drawing, typically a candidate *output* drawing. For notational convenience, we sometimes write positions as a function of this interpolation progress $0 \leq t \leq 1$, for example the position of a node u:

$$u(t) = \begin{pmatrix} (1-t) \cdot X_u' + t \cdot x_u \\ (1-t) \cdot Y_u' + t \cdot y_u \end{pmatrix}$$

In particular, we use the notation $\mathcal{D}(t)$ for the drawing where every node is linearly interpolated from the *current* drawing to the drawing $\mathcal{D}$.

In order to specify which part of the network should be enlarged as focus area, we take as input a set of *focus nodes* $F \subseteq V$ and a zoom factor Z. For each node $u \in F$ we fix $s_u = Z$.

We will now state the minimum-distortion focus map as the least-squares solution of an overconstrained system of linear equations. By moving the nodes of the network, such

a solution may introduce edge crossings that were not present in the input, thereby changing the apparent topology of the network. This is something we do not tolerate. Guaranteeing this is actually the complicated part of the algorithm. We handle this in Section 4 after we have stated the basic model: for now we disregard edge crossings.

3.2. Constraints

Recall that constraints in a least-squares optimization problem can be weighed: multiplying both the sides of a constraint by a constant increases the weight of its error term. We associate a weight with every type of constraint.

Our nominal objective is to minimize distortion. To achieve this we have soft constraints that say there should be no distortion at all. This is effectively the same approach as in the quadratic program of Haunert and Sering (2011) since we compute least-squares violation of the constraints. Note, however, that we will add more constraints.

This leads to the following constraint for the x component (Equation (4)) and y component (analogous) separately, where W_{distort} is a weight for 'distortion' constraints.

$$\forall u \in V,\ u \in N(u) : \frac{W_{\text{distort}}}{Dist(u,v)}\left[(x_v - x_u) - s_u(X_v - X_u)\right] = 0 \tag{4}$$

This corresponds to the definition of distortion in Equation (3). For focus nodes $v \in F$, we do not use the variable s_u but instead substitute the constant Z. This means we effectively have the hard constraint that $s_u = Z$.

Next we have a version of the bounding box constraint: we need to constrain the output drawing to some allowed region. If there were no such constraint, the optimum solution would be to scale everything uniformly (which has no distortion), and our problem would be easy and its solution useless indeed: that is not a focus map. We choose the bounding square of the original drawing as our allowed output region.

Note, however, that our framework does not allow us to express inequality constraints. Our solution for this is to constrain nodes that already lie near the border: we constrain their movement to be parallel to the boundary that they are near. That is, if a node is close to a vertical border, we have a constraint that says its x coordinate should not change. To this end, let B_X and B_Y be the sets of nodes that lie at most some fixed distance away from a border and should therefore have their x respectively their y coordinate constrained. (Note that a node may be in both B_X and B_Y. Then it is constrained not to move at all.) It may depend on the input map which sets B_X and B_Y work best. We have good experiences with a distance cutoff in the range of about 5% to as much as 20% of bounding-box side length. See Figure 3.

$$\begin{aligned} \forall u \in B_X :&\quad W_{\text{border}} \cdot (x_u - X_u) = 0 \\ \forall u \in B_Y :&\quad W_{\text{border}} \cdot (y_u - Y_u) = 0 \end{aligned} \tag{5}$$

We consider that we may have a 'preferred location' for the focus nodes: otherwise the focus area might move far away from its current position, which can be undesirable in an interactive application or for design purposes. Let (P_u^x, P_u^y) be this preferred location for node u. One might pick the preferred locations, for example, such that the center of gravity of the focus nodes does not change. In any case, the constraint is as follows.

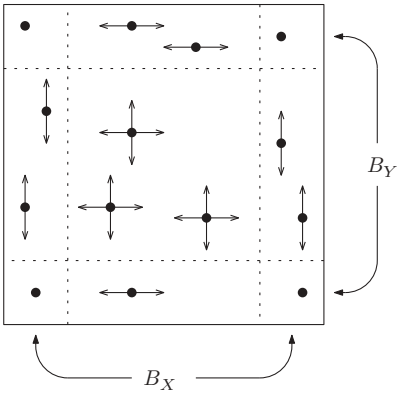


Figure 3. The movement of nodes near the border of the viewing window is constrained by Equation (5): they should only move parallel to it. This fixes nodes that lie near a corner.

$$\begin{aligned} \forall u \in F: \quad & W_{\text{place}} \cdot (x_u - P_u^x) = 0 \\ \forall u \in F: \quad & W_{\text{place}} \cdot (y_u - P_u^y) = 0 \end{aligned} \tag{6}$$

We introduce a further constraint in order to handle the needs of a system with user interaction.

Solving the above model with only slightly different focus nodes might give a very different output drawing. This behavior is typical of optimization approaches: the actual optimum solution may change drastically from one input to the next, even when inputs are very similar. This can be confusing for the user in an interactive application. We therefore introduce a kind of stabilization. We can do this using a constraint similar to Equation (6), but now with the *current* location of the node as its preferred location. (Recall the three drawings as in Figure 2.)

$$\begin{aligned} \forall u \in F: \quad & W_{\text{move}} \cdot (x_u - X_u') = 0 \\ \forall u \in F: \quad & W_{\text{move}} \cdot (y_u - Y_u') = 0 \end{aligned} \tag{7}$$

We use one further kind of constraint, which we call an *event constraint*. It is, in a sense, a 'private' constraint used by the algorithm: it does not directly model something we want to optimize but instead is used by the algorithm to ensure that no edge crossings are introduced. This constraint is given by Equation (11) in Section 4, where we describe how to handle crossings.

3.3. Solving the model

Let A be the matrix arising from the above set of constraints. Since the columns in this matrix correspond to the variables in our set of equations, there are at most $3|V|$ columns: x, y, and s for every node, where some s_u may be fixed. The number of rows in the matrix equals the number of constraints. There is a constraint for every neighbor of every node, and some additional constraints per node. The model consists of these $\mathcal{O}(|E|)$ constraints mentioned above and may have a further $\mathcal{O}(|V|)$ *event* constraints. In practice the number of event constraints is usually very small, certainly much smaller than $|V|$.

Since every row in this matrix corresponds to a constraint, this matrix is easily seen to be row-sparse: every constraint involves only a constant number of variables. Under reasonable assumptions, the matrix is also column-sparse. Consider the x_u variable for some node u.

- The number of distortion constraints (Equation (4)) it is involved in is exactly 2 deg(u): once in each direction for each incident edge.
- It occurs in exactly one movement constraint (Equation (7)).
- It occurs at most once in a border constraint (Equation (5)) and at most once in a placement constraint (Equation (6)).

For road networks it is reasonable to assume bounded degree, which then directly bounds the number of occurrences in these constraints. The occurrences of the variables y_u and s_u are similarly bounded. Lastly, the variables can be involved in some amount of event constraints. Many variables will not be involved in any events at all; others might be involved in some small amount of events. In our experiments on the realistic instances in Section 7, the average number of nonzeros per column was about 5.

In order to find a least-squares solution to our set of equations, we construct the corresponding matrix A and vector b. Using a sparse representation of A, we compute $A^{\mathrm{T}}A$ and $A^{\mathrm{T}}b$ and then solve $A^{\mathrm{T}}Ax^* = A^{\mathrm{T}}b$. The output drawing can be read from x^*. We note that, importantly, also $A^{\mathrm{T}}A$ is very sparse in our experiments, with an average of about seven nonzeros per row and per column. Our implementation uses the Eigen library (Guennebaud *et al.* 2010) for all matrix computations.

At this point we should emphasize again that none of the constraints are 'hard' constraints, that is, any constraint may be violated. What we get from the (black-box) solver is a solution with least-squares violation. This means that any time we solve a particular model, we cannot be sure that any specific equality included in the matrix holds. For example, the border constraints (5) do not *guarantee* that the output drawing is contained in our intended bounding box. This is something that we can handle using multiple iterations and can influence by picking appropriate weights.

4. Event constraints

We now describe our *event constraints*. These are used to ensure that the output drawing does not introduce edge crossings.

4.1. Lazy constraint generation

The focus maps of Haunert and Sering (2011) ensure that no edge crossings are introduced. Their approach, based on quadratic programming, is as follows. First a quadratic program, very similar to our model, is solved without consideration for edge crossings. Then the output is checked for edge crossings. If there are none, the algorithm is done. Otherwise, an inequality constraint is introduced for each existing crossing that disallows specifically this crossing (see Figure 4a). This *row generation* approach is effective in achieving a crossing-free drawing using only a relatively small amount of added constraints and in only a few iterations.

We follow a similar approach of lazily adding constraints to our model as we encounter edge crossings. It is, however, not immediately clear how to implement a non-crossing constraint using only equalities: it seems a non-crossing constraint is most naturally expressed as an inequality, but in our framework we cannot express inequalities.

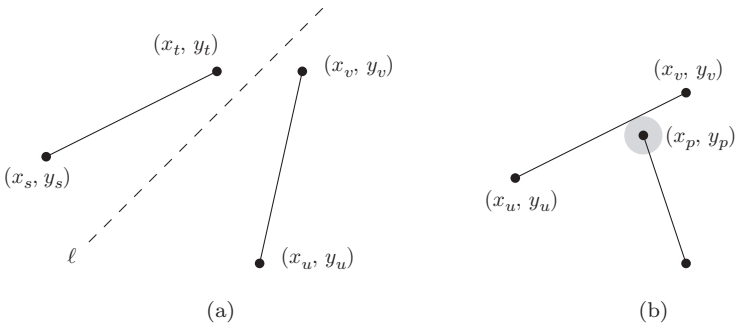


Figure 4. Comparison of non-crossing constraints. (a) Non-crossing constraint in the quadratic-programming approach of Haunert and Sering (2011). If two edges and found to cross in a candidate solution, an inequality constraint is added: the involved line segments need to be separated by a line of some given slope. (b) Non-crossing constraint based on an event. Consider the highest value of t such that $D(t)$ is crossing free. Then some node p lies on some edge uv. A soft equality constraint is added: the point p should remain at this relative position between u and v, with a little offset.

Using only equality constraints allows us to find least-squares solutions much faster. It is in making this trade-off that our algorithm achieves better performance. We use an approach based on what we call 'events.'

4.2. Events

Recall that our algorithm considers multiple drawings of the same map simultaneously. In particular, there is a *current* drawing of the map, and after we calculate a least-squares solution to our model, we have a candidate *output* drawing. If the output drawing is crossing free, we are done. Otherwise, consider the interpolation from the current drawing to the output drawing: $\mathcal{D}(\tau)$ for $0 \leq \tau \leq 1$.

As a precondition of the algorithm, the $\mathcal{D}(0)$ is crossing free. (The original input drawing is required to be crossing free, and the algorithm only accepts an output drawing if it is crossing free.) Then if $\mathcal{D}(1)$ is not crossing free, there must be some latest time $\tau < 1$ such that $\mathcal{D}(\tau)$ is crossing free. See Figure 5. At this time, a node touches a segment: in the movement of the interpolation, it is about to move through that segment. At the exact time the node hits the line segment, we have the following equality, where τ is the interpolation time, p the node, and (u, v) the segment. There exists an $\alpha \in [0,1]$ such that

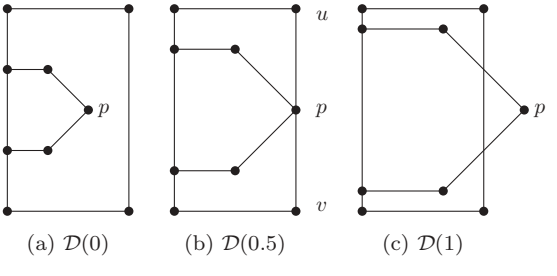


Figure 5. Example illustrating the existence of event times. Here $\mathcal{D}(0)$ is crossing-free and $\mathcal{D}(1)$ has crossings. The drawing $\mathcal{D}(0.5)$ is the last that has no (proper) crossings, at which time the node p is about to cross the edge $\{u, v\}$ and Equation (8) holds: $p(0.5) = (1–\alpha)\ u(0.5) + \alpha \cdot v(0.5)$ for some α.

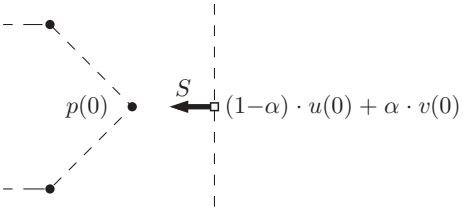


Figure 6. The offset vector S corresponding to the event in Figure 5. In Figure 5b it is determined that an event is needed for node p and edge $\{u, v\}$, with some parameter α. The value of S is based on the original drawing, that is, Figure 5a.

$$p(\tau) = (1 - \alpha) \cdot u(\tau) + \alpha \cdot v(\tau) \tag{8}$$

In this way, an event is defined by the five-tuple (p, u, v, τ, α). In order to get rid of this crossing (imminent at time τ), we could introduce a constraint to the model: we *enforce* Equation (8) at time 1. Such a constraint will stop this node from moving through this edge. In order to leave some space and also to have some safety when this soft constraint is somewhat violated, we add some displacement S. That is, we add the constraint

$$p = (1 - \alpha) \cdot u + \alpha \cdot v + S \tag{9}$$

where

$$S = \frac{P' - [(1 - \alpha) \cdot U' + \alpha \cdot V']}{2} \tag{10}$$

The vector S pushes the desired position for p toward p's original position: from the intersection point on the current positions of $\{u, v\}$ to the current position of p, scaled down by a factor 2. This is an easy and robust way to always get a reasonable offset. See Figure 6 for an illustration. Note that S is a constant since α is fixed for any specific event. The constraint is then as follows, where $(S_x, S_y) = S$ as above.

$$\begin{aligned} W_{\text{event}} \cdot [(1 - \alpha)x_u + \alpha x_v - x_p - S_x] &= 0 \\ W_{\text{event}} \cdot [(1 - \alpha)y_u + \alpha y_v - y_p - S_y] &= 0 \end{aligned} \tag{11}$$

In this way, we glue a node to a segment, preventing the crossing. See Figure 4b for an example.

5. Algorithm

The high-level overview of the algorithm is as follows. We start with the model from Section 3, with no event constraints. We find the least-squares drawing $\mathcal{D}$ and check it for edge crossings. If there are none, we are done since we have found a crossing-free drawing that is optimal given the current model. Otherwise we add event constraints in order to get rid of the edge crossings. Since these are not just inequality constraints, but actually glue a point onto a segment, we should be careful here: the actual intersections that are present in $\mathcal{D}(1)$ at this point may not be good events to add. Consider for example Figure 7, where it would be a bad idea to glue $\overline{uv}$ to the topmost line segment.

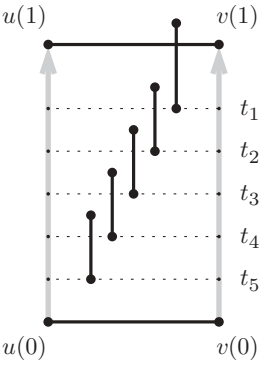


Figure 7. Consider the case where the horizontal edge $\overline{uv}$ moves as indicated.

The basic idea is to find the latest crossing-free drawing when interpolating linearly from the current drawing to $\mathcal{D}$. We would add an event constraint for the crossing that is about to happen and then this crossing will not happen if we solve the model again. Finding this event can be done in $\mathcal{O}(nm)$ time (enumerate all pairs of a node and a segment), but that is too slow for our purposes. We use the following algorithm that works well in practice.

Let C be the set of crossings in $\mathcal{D}$, where a crossing is a pair of edges. Now we find the earliest t at which any of these crossings occur in $\mathcal{D}(t)$: before this time, the crossings in C do not occur. We call finding this time a *rewind* and, given C it can be done in $\mathcal{O}(|C|)$ time: considering a single crossing takes constant time. Still, $\mathcal{D}(t)$ may have other crossings. So we repeat this process until we have a crossing-free drawing. (In Section 7 we show experimentally that this process terminates in a few steps.) Then we add event constraints for some of the earliest intersections that were going to occur.

After selecting new event constraints, we find a least-squares drawing again (with added constraints) and repeat the above process until we find a drawing where $\mathcal{D}(1)$ has no crossings. Then we are done. A detailed explanation of the algorithm follows.

5.1. Outer loop

In this description of the algorithm, we consider a global 'model.' This starts out as the model from Section 3. During the algorithm, we may add event constraints to this model in order to get rid of edge crossings.

Algorithm 1: OuterLoop.

```
done ← false
while -done do
    done ← Step()
```

Algorithm 1 is the 'outer loop' of the algorithm. We just keep calling Step, which either finds a crossing-free drawing or adds some event constraints to the model.

5.2. Step

In Algorithm 2 we calculate the least-squares solution to the model and check whether this drawing has edge crossings. It is crucial for the performance of the algorithm that sparse

matrix representations are used. Checking for edge crossings can for example be done using the $\mathcal{O}(n \log n + k)$ time sweepline algorithm for line segment intersection (see for example De Berg *et al.* 2008), where k is the number of crossings found. If there are crossings, we initiate the 'rewinding' described before. At the end of this, we add some event constraints to the model.

Algorithm 2: Step.

1: Drawing $\mathcal{D} \leftarrow$ Least-squares solution to model
2: $t \leftarrow 1$
3: *Events* $\leftarrow \emptyset$
4: $C \leftarrow$ Crossings($\mathcal{D}(t)$)
5: **while** $|C| \geq 1$ **do**
6: $\quad$ *t, Events* $\leftarrow$ Rewind($\mathcal{D}$, C)
7: $\quad$ $C \leftarrow$ Crossings($\mathcal{D}(t)$)
8: Add *Events* to model
9: **return** $t = 1$

In general, the while loop in line 5 of Algorithm 2 can execute $\Omega(n)$ iterations: consider again Figure 7. The first iteration will find time t_1. In the subsequent iterations, each edge will be touched. In practice, we observe that a small number of iterations usually suffices. See our experimental results in Section 7.

5.3. *Rewind*

The rewind procedure (Algorithm 3) gets as input the drawing D and a set C of pairs of edges. Every pair of edges $\{a, b\}$ and $\{c, d\}$ in C has the property that it crosses in $\mathcal{D}(t)$ for some $0 \leq t \leq 1$. The procedure does not need to know this value t.

By precondition of the algorithm, we know that $\mathcal{D}(0)$ is crossing free. Then there is an earliest time at which a pair of edges in C cross. At that time, Equation (8) holds: a or b lies on the line segment $\overline{cd}$, or c or d lies on the line segment $\overline{ab}$. We test these four possibilities for each pair in C and collect all these events.

Let $t_{\min}$ be the smallest time among these events. Then, by construction, none of the crossings in C occur in $\mathcal{D}(t_{\min})$. Maybe some other edges now cross but none of the pairs in C. We return the event at time $t_{\min}$ to be added to the model. As an optimization, we also return all other events that occur soon after. We have used a cutoff of $2 \cdot t_{\min}$. This is a trade-off between returning fewer events, possibly requiring more iterations of outerloop, and returning more events, possibly adding events to the model that are unnecessary.

Algorithm 3: Rewind($\mathcal{D}$, C).

1: *Events* $\leftarrow \emptyset$
2: **for all** $\{\{a, b\}, \{c, d\}\} \in$ C **do**
3: $\quad$ **if** CheckForEvent(a,c,d) occurs in D **then** Add the event to *Events*
4: $\quad$ **if** CheckForEvent(b,c,d) occurs in D **then** Add the event to *Events*
5: $\quad$ **if** CheckForEvent(c,a,b) occurs in D **then** Add the event to *Events*
6: $\quad$ **if** CheckForEvent(d,a,b) occurs in D **then** Add the event to *Events*
7: $t_{\min} \leftarrow$ minimum time in *Events*
8: **return** $t_{\min}$, and all events in *Events* with $\tau \leq 2 \cdot t_{\min}$

5.4. Check for event

This procedure, called by REWIND, determines whether point p and edge $\{u, v\}$ satisfy Equation (8) for some $0 \leq \alpha \leq 1$ and $0 \leq \tau \leq 1$. This calculation can be made by solving a quadratic equation in τ.

6. Improvements for use in interactive systems

We present two improvements to the algorithm that are mainly useful in an interactive system. The first is a way to display partial progress while the algorithm is still computing. The second is a heuristic to improve the runtime of the algorithm in a common use case by reusing partial results from earlier runs.

6.1. Showing progress

After finishing the while loop on line 5 in Algorithm 2 we have a drawing D and a time t such that $\mathcal{D}(t)$ is crossing free. This drawing can be displayed to the user. Over the course of the various steps of the algorithm, this allows us to show progress toward the final drawing. This greatly improves the responsiveness of an interactive system. Section 7 provides experimental evaluation of the rate at which the algorithm provides such progress.

The various iterations of the algorithm introduce constraints. These constraints influence the new candidate drawing D, and this new drawing may have crossings that were not present before. It is possible that a later iteration, under the influence of these new constraints, yields a lower value of t than an earlier iteration. This is not a problem for the algorithm but presents an animation that sometimes seems to jump back. We therefore only display the new drawing if its value of t is an improvement over what is currently displayed.

6.2. Remembering events

Consider the situation where we have already run the algorithm with a certain set of focus nodes F_1. In an interactive system it could be a common scenario that the algorithm is later run again with a very similar set of focus nodes F_2. The algorithm is likely to find many of the same, or at least similar, event constraints in both runs. We can start the algorithm with the set of event constraints found in the previous run of the algorithm, which has two consequences. On the one hand, there is an influence on the runtime, because we reuse work from a previous run. On the other hand, there is an influence on the quality of the solution, because the algorithm may end up using a different set of constraints. This set of constraints might be worse, since they were not specifically calculated for F_2. In our experiments we found that this procedure is quite effective, somewhat dependent on the input network: the average runtime improves by approximately 25–50%, at the cost of typically a few percent increase in distortion.

7. Experiments

We have implemented this algorithm, including the improvements from Section 6, in C++. We use CGAL's 'intersection of curves' package (Zukerman *et al.* 2013) to check drawings for edge crossings and use Eigen (Guennebaud *et al.* 2010) for sparse linear algebra. All experiments have been run on a desktop PC with an Intel® Core™ i5-2400

CPU at 3.10 GHz. Memory usage is not an issue in these experiments, since the sparse representation of the matrices involved does not even exceed 1 MB in size. (Dense representation would take several hundred megabytes.) Unless otherwise noted, we use scale factor $Z = 2$.

7.1. Basic inputs

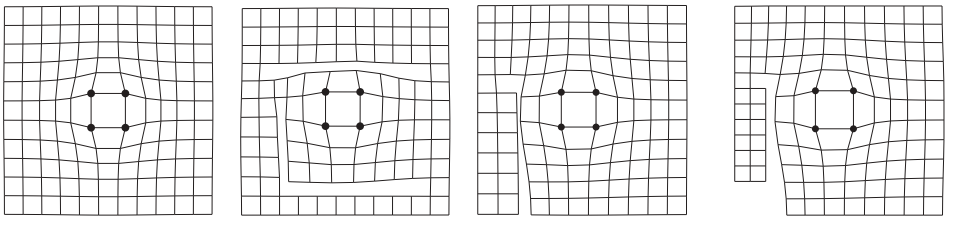
In Figure 8 we review the results of our algorithm on some basic instances from Haunert and Sering (2011): a full grid and some subsets of grids. Our drawings look similar and have a similarly low amount of distortion. Apart from this, we notice one thing in particular about Figure 8c and d, where we show our solution and the quadratic programming approach on the same instance: this is the only instance in which our solution is markedly different from the quadratic programming approach. This difference comes from the handling of the bounding-window constraint: with our boundary constraints the bottom-left nodes are in fact constrained to remain near the bottom. In this case the difference in measured distortion is not large (an additional 13%), but we point out that this difference may lead to qualitatively different drawings.

7.2. Experimental setup

Our experiments consist primarily of two consecutive runs of the algorithm: a *cold start* and a *running start*. The cold start represents an offline computation or, in an interactive system, the first computation on a new map. The running start represents an interactive system where the user moves the focus area from one location to a nearby location, perhaps by dragging the mouse.

In order to be able to experimentally compare maps of different size and scale, we measure distances in 'screen space' as follows. Consider the size of the smallest square enclosing the input network: we call its side length 100%. In these experiments, when we pick a set F of focus nodes, we pick the nodes in some disc of radius 5%. When we move the center of the focus area, we move it by a distance 10%.

- **Cold start.** Pick a node v_1 of the network, uniformly at random. Then pick all nodes within a disc of radius 5% around v_1 and call this set F_1. Run the algorithm with focus nodes F_1. Remember the set of event constraints that the algorithm finds.



(a) (b) (c) Distortion: 30.55 (d) Distortion: 26.96 (Haunert and Sering 2011)

Figure 8. The result of our algorithm on several example networks (subsets of grids) with zoom factor $Z = 2$. The focus nodes F are indicated using dots. In subfigure (a) the network is a full grid. Subfigures (b) and (c) are run on different subsets of the grid. Subfigure (d), for comparison, is the quadratic-programming solution of Haunert and Sering (2011) for the same network as (c).

- **Running start.** Pick a random direction and, starting from the original position of v_1, move 10% in that direction: call the resulting point p. (Pick the direction uniformly from those directions that give a point p within the bounding square: it would not be reasonable for this experiment to move the focus outside of the map.) Let v_2 be the network node with original position closest to p. Then pick all nodes within a disc of radius 5% around v_2 and call this set F_2. Run the algorithm again, with focus nodes F_2, starting with the event constraints from the cold-start run already in the model.
- **Control run.** Run the algorithm again with focus nodes F_2, but starting without any event constraints. That is, the control run is a cold start on F_2.

By comparing the cold start to the running start, we can evaluate the effect on the runtime. By comparing the running start to the control, we can evaluate the effect on the output. The results in this section are for 1000 runs of this experiment on each map.

7.3. Realistic inputs

Now we evaluate our algorithm on several networks that we consider reasonable inputs. These networks are listed in Figure 9. First we look at some of the results of the algorithm. Figure 10 shows several drawings of Wuerzburg, where in sequence various focus areas are added to the map. (The algorithm is not limited to a single 'focus area': the set F of focus nodes can be chosen arbitrarily.) These could be, for example, several steps in a sequence of interactions with a system for designing a route map or tourist map, where important navigation actions or points of interest are enlarged for legibility and focus. This system would react to new focus areas in real time and provide a smooth animation from the current drawing to the newly optimized drawing including a further focus area. A video of this interaction with our implementation of the algorithm is available as supplemental material. Figure 11 shows several more examples on the other input maps. The results are similar to those of the quadratic programming approach, but, as we will discuss next, the runtime is much improved.

The results of our runtime measurements are given in Figures 12 (Wuerzburg), 13 (Boston), and 14 (Britain). In these, subfigure (a) shows a histogram of runtimes for 1000 repetitions of the experiment. We can generally observe that the runtime of a cold start is

(a) Wuerzburg (b) Boston (c) Britain

Figure 9. The input maps for the experiments in Section 7.

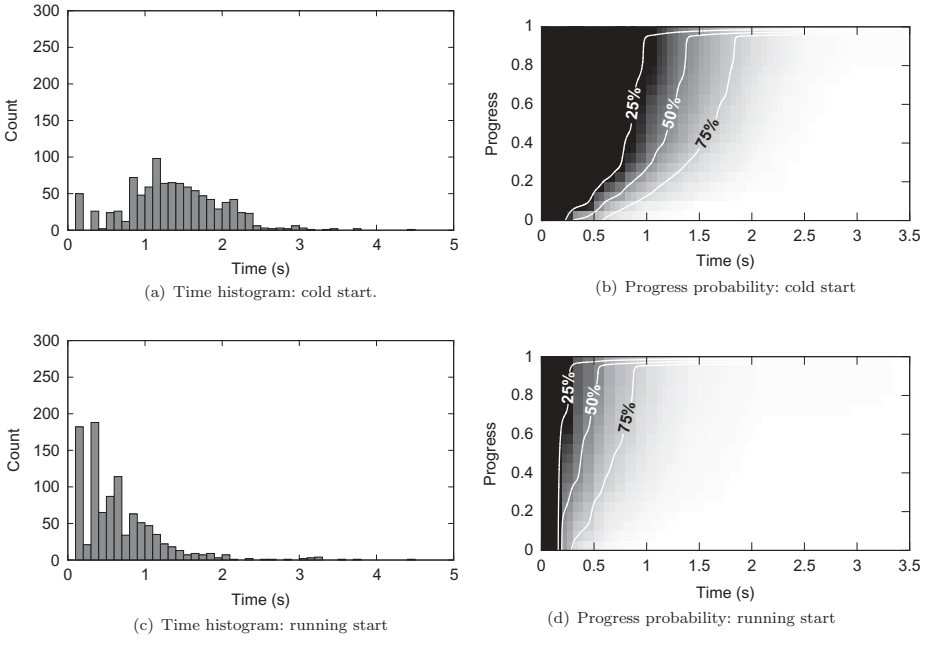
Figure 10. Several simultaneous focus areas in Wuerzburg, with $Z = 2$.

Figure 11. Several focus areas in Boston and Britain, with $Z = 3$.

reasonable for an interactive system, with mean runtime well below 1 second on two of the three maps tested. This is much improved over the quadratic programming approach of Haunert and Sering (2011) who report runtimes of 4.33 s and 7.08 s on Boston and Wuerzburg, respectively.

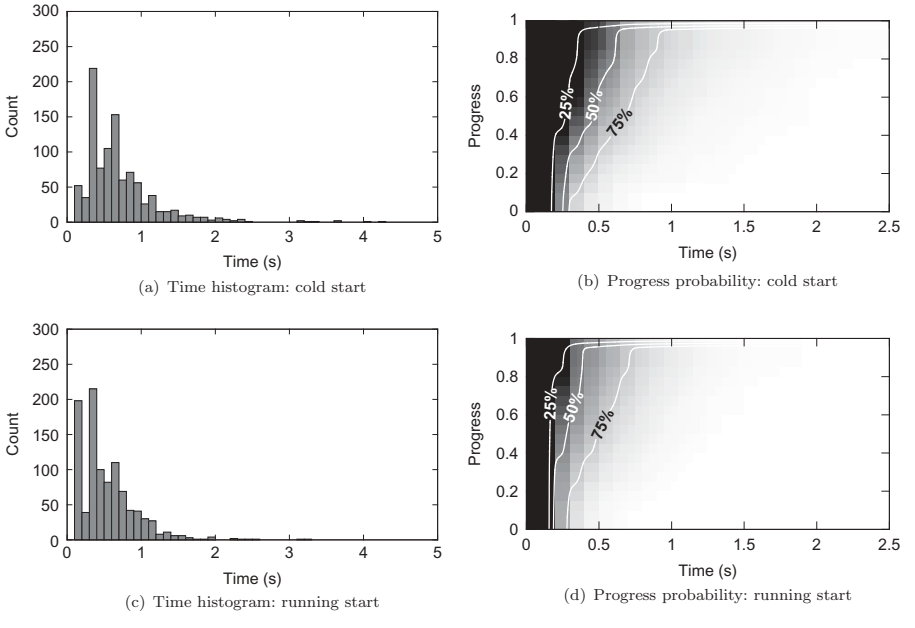
Name	$\|V\|$	$\|E\|$	Description
Wuerzburg	2511	2995	OpenStreetMap road network of Würzburg http://download.geofabrik.de/osm/
Boston	2821	3332	MassGIS road network of Boston http://www.mass.gov/mgis/eotroads.htm
Britain	3112	3240	Eurostat map of NUTS-3 regions in Britain http://epp.eurostat.ec.europa.eu/cache/GISCO/geodatafiles/NUTS_2010_10M_SH.zip

Still, waiting for the entire computation to complete could be problematic for a smoothly animating focus area and is also an annoying delay in an interactive system. This is where the improvements from Section 6 come in. In subfigure (c) we can see the effect of the



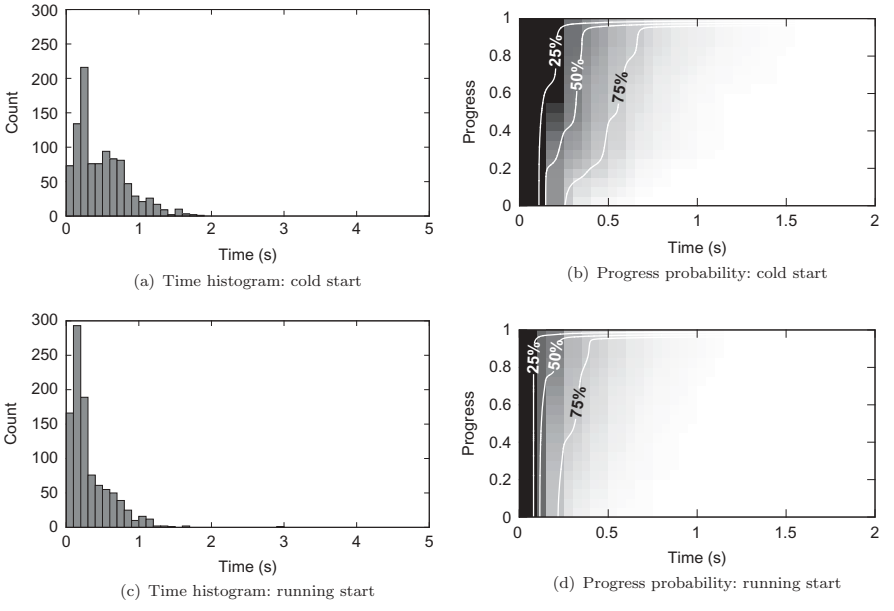
(a) Time histogram: cold start.
(b) Progress probability: cold start
(c) Time histogram: running start
(d) Progress probability: running start

Figure 12. Runtime statistics for Wuerzburg, based on 1000 runs. A running start reduces the mean runtime from 1.37 s to 0.65 s, a reduction to 47%.



(a) Time histogram: cold start
(b) Progress probability: cold start
(c) Time histogram: running start
(d) Progress probability: running start

Figure 13. Runtime statistics for Boston, based on 1000 runs. A running start reduces the mean runtime from 0.69 s to 0.52 s, a reduction to 75%.



(a) Time histogram: cold start

(b) Progress probability: cold start

(c) Time histogram: running start

(d) Progress probability: running start

Figure 14. Runtime statistics for Britain, based on 1000 runs. A running start reduces the mean runtime from 0.49 s to 0.32 s, a reduction to 65%.

running start. Furthermore, the algorithm can already provide partial progress during the run of the algorithm. In this way, an animation from input to result can be started much sooner than suggested by the histograms and the mean runtime.

This is visualized in subfigures (b) and (d), which we will discuss now.

A step of the algorithm finds a drawing D and an interpolation parameter p such that $\mathcal{D}(p)$ is crossing free. This interpolated drawing is the one that is presented to the user. We call p the *progress* of the algorithm, since $p = 0$ corresponds to the input and $p = 1$ corresponds to the final output. Then we can ask the following probabilistic question about the performance of the algorithm: after some time t, what is the probability that at least progress p has been made? We have experimentally determined this probability, again using 1000 repetitions of the experiment. This is shown in the subfigures (b) and (d), where black is a low probability (none of the thousand runs were this fast) and white is high probability (all thousand runs were at least this fast). We show slightly smoothed contour lines for 25%, 50%, and 75%.

The top of these diagrams ($p = 1$) represents waiting until the algorithm is finished, and this corresponds to the histograms in subfigures (a) and (c). These data indicate that, quite confidently, an animation can already be started well before the algorithm is finished. If this animation is started with a small initial delay and at a reasonable speed, the progress of the algorithm is highly likely to stay ahead of the animation.

Now we discuss in more detail the memorization of event constraints. This is evaluated using the *running start* experiments, and the results can also be seen in Figures 12–14, where (a) and (b) are with a cold start, and (c) and (d) are with a running start. Notice the improvement of the runtime from start to finish and also that there is more progress, sooner during the run of the algorithm. We see from the data that the effectiveness of this running start varies from map to map: on Wuerzburg it is very

Table 1. Statistics of the runtime, based on 1000 runs.

	Wuerzburg	Boston	Britain
Mean			
Cold	1.37 s	0.69 s	0.49 s
Running	0.65 s	0.52 s	0.32 s
Coefficient of variation			
Cold	0.46	0.88	0.70
Running	0.84	0.72	0.88

effective, reducing the mean runtime to 47%, whereas the effect on Boston is relatively minor (reduction to 75%). The difference between the maps also expresses itself in the variance of the runtime: with a cold start on Wuerzburg, the coefficient of variation[1] is remarkably lower than any of the other experiments (see Table 1). This can be explained as follows.

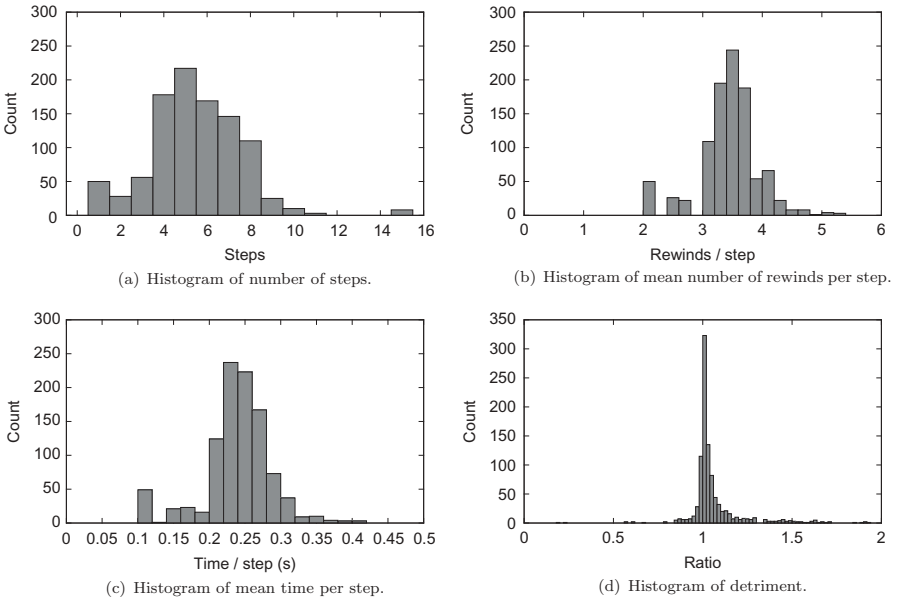
In every run of the experiment, the algorithm finds a set of constraints E_{cold} in the cold start run, and a set of constraints $E_{control}$ in the control run. We can measure the similarity of these two sets, for which we use the Jaccard similarity index. (For two sets A and B the Jaccard similarity index is defined as $|A \cap B| / |A \cup B|$.) We consider two event constraints to be equal if they involve the same node and the same edge, disregarding the value α. (If we would include α, a real number, it is unlikely that any two constraints are ever exactly equal.) A high similarity index would explain a big gain from a running start: if the cold start on F_1 and the control run on F_2 use similar constraints, then the running start on F_2 will be effective.

This is indeed the case here: in the 1000 runs, Wuerzburg yields a mean similarity of 0.24, while Boston yields mean similarity 0.11. This also explains the lower coefficient of variation for cold starts on Wuerzburg: every run spends some time finding, partly, a similar set of event constraints. The improvement in running time from a running start on Britain (reduction to 65%) is between that of the other two maps, and indeed it yields mean similarity 0.16.

Having established that the running start is beneficial for the runtime of the algorithm, we check what the effect is on the quality of the output. To this end we can compare the distortion in the running start with the distortion in the control run: both are for the same set of focus nodes F_2. Dividing the former by the latter gives a measure of how much distortion is added: call this ratio the *detriment*. Figure 15d shows a histogram of detriment on Wuerzburg (results on the other maps are similar). We see that it is concentrated slightly above 1, with first quartile 1.00 and third quartile 1.07. That is, the running start typically comes with at most a 7% increase in distortion, with some chance of a worse result. Note that we sometimes observe detriment slightly less than 1: improvement. This is possible because the generation of event constraints is heuristic, and the running start might get lucky because of the different initialization.

After a running start, if there is not an immediate request for yet another focus area, an interactive system can itself do the 'control' run as a kind of cleanup step. In this way, the fast response time of the running start can be combined, eventually, with the quality of a cold start.

Two important aspects in the runtime of the algorithm are the number of *steps* (Algorithm 2: finding new event constraints) and the number of *rewinds* (Algorithm 3:



(a) Histogram of number of steps.
(b) Histogram of mean number of rewinds per step.
(c) Histogram of mean time per step.
(d) Histogram of detriment.

Figure 15. Further statistics on Wuerzburg, based on 1000 runs. Figures (a), (b), and (c) are for a cold start. Figure (d), since it is about detriment, compares the running start to the control run.

finding a crossing-free time). Here we only provide figures for Wuerzburg, the 'hardest' instance for our algorithm: results on the other maps are similar in distribution.

Figure 15a shows a histogram of the number of steps performed by the algorithm. Generally, the number of steps required is quite low, with a mean of 5.47. In Figure 15b we show a histogram of the average number of rewinds per step, where averages are taken for every run. As we noted in the description of the algorithm, this can be as much as $\Omega(n)$ in the worst case, but in practice the procedure works well, with a mean of 3.41 rewinds per step (coefficient of variation 0.15).

Figure 15c shows a histogram of the average time spent on a step. This is a meaningful quantity as it is the time the algorithm requires to provide a new partial-progress drawing that can be presented to the user. On Wuerzburg, our hardest instance, it has mean 0.24 s with a standard deviation of 0.04 s. This means the algorithm can reliably provide an animation system with about four keyframes per second. If this animation system shows a linear interpolation between the keyframes, the animation can start with a delay of only one keyframe, that is, about 0.25 s.

8. Conclusion

We have presented a new algorithm that enlarges a focus region in a given network map without removing non-focus (i.e., context) network parts from the map or changing the map's size. Our algorithm achieves this goal with relatively little distortion. In fact, the results are similar to those produced with the method of Haunert and Sering (2011), who showed that the maps produced with their method are far less distorted than the maps generated with a classical fish-eye projection. In contrast to the method of Haunert and Sering (2011), our new algorithm has been designed for application in an interactive

system, in which a user can brush edges of a network map to set the focus. The network currently displayed in the map then continuously warps to the output of the algorithm. We think that interactivity is of ultimate importance for the type of focus maps we produce, in particular, because it can be difficult to recognize the distortion of such maps without additional visual cues. We argue that an animation mediating between the currently visible map and the output map helps a user understand what information can be inferred from the map displayed – and what cannot.

In order to achieve a performance that allows for interactivity, we have chosen an approach based on least-squares optimization. To this end, we had to replace hard inequality constraints in the model of Haunert and Sering (2011) with soft constraints. Those are represented with linear equations, whose degree of violation we measure in our objective function. We have given special consideration to the avoidance of unwanted edge crossings. Basically, we solve our model first without constraints avoiding edge crossings. The solution we find is then used to identify a small set of constraints needed for a crossing-free solution and, beyond this, allows us to start the animation before the final solution is found. Moreover, memorizing the non-crossing constraints from an initial run of our algorithm allows us to achieve a better runtime for any further run – assuming that the focus region does not move much between two consecutive runs.

We have evaluated the runtime of our algorithm on three networks of several thousand nodes and edges with random focus regions. While the method of Haunert and Sering (2011) needed approximately 7 s for the hardest instance (Wuerzburg), our new algorithm needed 1.37 s on average if not given information from a previous run. Memorizing the non-crossing constraints from a previous run with a similar focus reduced the average runtime to 0.65 s. Even without this information, our algorithm was able to begin with the animation after 0.25 s on average. We conclude that the response time of our algorithm is well below 1 s, thus it allows for real-time interaction.

An interesting question for future research is how to combine our method with map generalization, for example, selection or simplification. In order to convey the distortion of a static focus map to the user, we already have developed a schematization algorithm (Van Dijk *et al.* 2013). What is missing, however, are real-time generalization algorithms that can be applied in conjunction with our interactive method. We are also well aware of the fact that a network alone does not make a map, so we plan to add objects like salient landmarks and labels to the map. How to consider such objects when searching for a good network layout is an open problem that we plan to address.

Funding

This research was supported by *Algorithms for Interactive Variable-scale Maps* of the German Research Foundation (DFG) [grant number 5451/3-1] and carried out in the context of the Internet Research Center (IRC) at University of Würzburg.

Note

1. The coefficient of variation is defined as standard deviation divided by mean.

References

Agrawala, M. and Stolte, C. 2001. Rendering effective route maps: improving usability through generalization. *In*: *Proceedings of the 28th annual conference on computer graphics and interactive techniques (SIGGRAPH'01)*, Los Angeles, CA. New York: ACM, 241–249.

Card, S.K., Moran, T.P., and Newell, A., 1980. The keystroke-level model for user performance time with interactive systems. *Communications of the ACM*, 23 (7), 396–410. doi:10.1145/358886.358895

Cecconi, A. and Galanda, M., 2002. Adaptive zooming in web cartography. *Computer Graphics Forum*, 21 (4), 787–799. doi:10.1111/1467-8659.00636

De Berg, M., *et al*., 2008. *Computational geometry: algorithms and applications*. Berlin: Springer.

Guennebaud, G., *et al*., 2010. Eigen v3. [online]. Available from: http://eigen.tuxfamily.org [Accessed 2 July 2013].

Hampe, M., Sester, M., and Harrie, L. 2004. Multiple representation databases to support visualization on mobile devices. *In*: M.O. Altan, ed. *Proceedings of the 20th congress of the international society for photogrammetry and remote sensing (ISPRS)*, 12–23 July, Istanbul, 135–140.

Harrie, L., Sarjakoski, L.T., and Lehto, L., 2002. A mapping function for variable-scale maps in small-display cartography. *Journal of Geospatial Engineering,* 4 (2), 111–123.

Harrie, L. and Sarjakoski, T., 2002. Simultaneous graphic generalization of vector data sets. *Geoinformatica,* 6 (3), 233–261. doi:10.1023/A:1019765902987

Haunert, J.H. and Sering, L., 2011. Drawing road networks with focus regions. *IEEE Transactions on Visualization and Computer Graphics,* 17 (12), 2555–2562. doi:10.1109/TVCG.2011.191

Haunold, P. and Kuhn, W. 1994. A keystroke level analysis of a graphics application: manual map digitizing. *In*: Beth, A., Susan T. Dumais, and Judith S. Olson, eds. *Proceedings of the conference on human factors in computing systems (SIGCHI)*, 24–28 April, Boston, MA. New York: ACM, 337–343.

House, D.H. and Kocmoud, C.J., 1998. Continuous cartogram construction. *Proceedings Visualization,* 98, 197–204.

Inoue, R. and Shimizu, E., 2006. A new algorithm for continuous area cartogram construction with triangulation of regions and restriction on bearing changes of edges. *Cartography and Geographic Information Science,* 33 (2), 115–125. doi:10.1559/152304006777681698

Jenny, B., 2006. Geometric distortion of schematic network maps. *Bulletin of the Society of Cartographers,* 40 (1 and 2), 15–18.

Jiang, B. and Claramunt, C., 2004. A Structural approach to the model generalization of an urban street network. *Geoinformatica,* 8 (2), 157–171. doi:10.1023/B:GEIN.0000017746.44824.70

Kopf, J., *et al.*, 2010. Automatic generation of destination maps. *ACM Transactions on Graphics,* 29 (6), 158:1–158:12. doi:10.1145/1882261.1866184

Kraus, K., 2004. *Photogrammetry – geometry from images and laser scans*. Berlin: de Gruyter.

Li, Q., 2009. Variable-scale representation of road networks on small mobile devices. *Computers & Geosciences,* 35 (11), 2185–2190. doi:10.1016/j.cageo.2008.12.009

Merrick, D. and Gudmundsson, J. 2006. Increasing the readability of graph drawings with centrality-based scaling. *In*: K. Misue, K. Sugiyama, and J. Tanaka, eds. *Proceedings of the 2006 Asia-Pacific symposium on information visualisation (APVis '06)*, Vol. 60, 1–3 February, Tokyo. Darlinghurst: Australian Computer Society, 67–76.

Miller, R.B. 1968. Response time in man-computer conversational transactions. *In*: *Proceedings of the December 9–11, 1968, fall joint computer conference, part I*, 9–11 December, San Francisco, CA. New York: ACM, 267–277.

Nollenburg, M. and Wolff, A., 2011. Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE Transactions on Visualization and Computer Graphics,* 17 (5), 626–641. doi:10.1109/TVCG.2010.81

Roth, R., 2013. Interactive maps: what we know and what we need to know. *Journal of Spatial Information Science,* 6, 59–115.

Schmid, F., 2008. Knowledge-based wayfinding maps for small display cartography. *Journal of Location Based Services,* 2 (1), 57–83. doi:10.1080/17489720802279544

Schmid, F., *et al.*, 2010. Situated local and global orientation in mobile you-are-here maps. *In*: *Proceedings of the 12th international conference on HCI with mobile devices and services*, 7–10 September, Lisboa. New York: ACM, 83–92.

Sester, M., 2005. Optimization approaches for generalization and data abstraction. *International Journal of Geographical Information Science,* 19 (8–9), 871–897.

Shimizu, E. and Inoue, R. 2003. Time-distance mapping: visualization of transportation level of service. *In*: *Proceedings of the symposium on environmental issues related to infrastructure development*, *JSPS Core university program on environmental engineering*, 8–9 August, Manila, 221–230.

Simonetto, P., *et al.*, 2011. ImPrEd: an improved force-directed algorithm that prevents nodes from crossing edges. *Computer Graphics Forum,* 30 (3), 1071–1080. doi:10.1111/j.1467-8659.2011.01956.x

Van Dijk, T.C. and Haunert, J.H. 2013. A probabilistic model for road selection in mobile maps. *In*: S. Liang, X. Wang, and C. Claramunt, eds. *Web and wireless geographical information systems., Vol. 7820 of lecture notes in computer science*. Berlin: Springer, 214–222.

Van Dijk, T.C., *et al*., 2013. Accentuating focus maps via partial schematization. *In*: *Proceedings of the 21st ACM SIGSPATIAL international conference on advances in geographic information systems (ACM SIGSPATIAL GIS 2013)*, 5–8 November, Orlando, FL. New York: ACM, 438–441.

Wang, Y.S. and Chi, M.T., 2011. Focus + context metro maps. *IEEE Transactions on Visualization and Computer Graphics,* 17 (12), 2528–2535. doi:10.1109/TVCG.2011.205

Wang, Y.S., Lee, T.Y., and Tai, C.L., 2008. Focus + context visualization with distortion minimization. *IEEE Transactions on Visualization and Computer Graphics,* 14 (6), 1731–1738. doi:10.1109/TVCG.2008.132

Wardlaw, J., 2010. Principles of interaction. *In*: M. Haklay, ed. *Interacting with geospatial technologies*. New York: Wiley, 179–198.

Yamamoto, D., Ozeki, S., and Takahashi, N. 2009. Focus + glue + context: an improved fisheye approach for web map services. *In*: *Proceedings of the 17th ACM SIGSPA-TIAL international conference on advances in geographic information systems (GIS '09) ACM*, 4–6 November, Seattle, WA. New York: ACM, 101–110.

Zukerman, B., Wein, R., and Fogel, E., 2013. 2D Intersection of curves. *In*: *CGAL user and reference manual*, 4, 2nd ed. CGAL Editorial Board.