Drawing Road Networks with Focus Regions

Jan-Henrik Haunert and Leon Sering

Fig. 1. A road map of Boston (left). A user selects a focus region (black circle) and sets a zoom factor Z, here $Z = 3$. Then, an output map (right) is generated. The focus region is scaled up by Z while the map still fits into its original frame. This is achieved by scaling some parts of the road network down that are not in the focus region. Distortions at road segments of the network are minimized.

Abstract—Mobile users of maps typically need detailed information about their surroundings plus some context information about remote places. In order to avoid that the map partly gets too dense, cartographers have designed mapping functions that enlarge a user-defined focus region – such functions are sometimes called fish-eye projections. The extra map space occupied by the enlarged focus region is compensated by distorting other parts of the map. We argue that, in a map showing a network of roads relevant to the user, distortion should preferably take place in those areas where the network is sparse. Therefore, we do not apply a predefined mapping function. Instead, we consider the road network as a graph whose edges are the road segments. We compute a new spatial mapping with a graph-based optimization approach, minimizing the square sum of distortions at edges. Our optimization method is based on a convex quadratic program (CQP); CQPs can be solved in polynomial time. Important requirements on the output map are expressed as linear inequalities. In particular, we show how to forbid edge crossings. We have implemented our method in a prototype tool. For instances of different sizes, our method generated output maps that were far less distorted than those generated with a predefined fish-eye projection. Future work is needed to automate the selection of roads relevant to the user. Furthermore, we aim at fast heuristics for application in real-time systems.

Index Terms—cartography, schematic maps, fish-eye view, graph drawing, optimization, quadratic programming.

1 Introduction

Variable-scale map projections have been frequently proposed for Internet cartography and mobile cartography. They yield a large-scale representation of a focus region (often the user's surrounding) and, thereby, allow many relevant details to be displayed. To avoid that the user looses context, also more remote regions are displayed on the same map. For two reasons, however, such regions are shown at small scale. First, by scaling remote regions down they become de-emphasized. Second, as map space is limited, not all information can be shown at large scale.

Using different scales on the same map implies distortions – if a user wants to measure distances, such a distorted map is useless. Many tasks, however, do not require the exact knowledge of distances. For example, in order to visualize driving instructions to a user, sketch-like maps with highly distorted distances, so-called *route maps*, are useful [1]. Other examples for maps with highly distorted distances are *metro maps* [24] and *destination maps* [17], which visualize how to reach a certain destination. More generally, we use the term *schematic map* for any map whose distortions result from some design principle applied and exceed those distortions commonly found in geographic maps – here distortions are mainly due to the projection of a sphere (the globe) onto the map plane and due to cartographic displacement.

Variable-scale maps are often designed with a function that maps each point of the plane (an object's position in a geographic map) to a second point (the object's position in the variable-scale map). Usually, some type of fish-eye view is chosen. It is difficult to decide, however,

• *The authors are with the Chair for Computer Science I, University of Würzburg, Am Hubland, D-97074 Würzburg, Germany. E-mail: jan.haunert@uni-wuerzburg.de, leon.sering@gmail.com.*

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.

whether a predefined mapping function is appropriate for a particular task and input. The existing functions ensure some basic properties of the output map, for example, they keep the focus region undistorted. They do not, however, optimize a well-defined function that measures the quality of the map. In contrast, in this paper we present an optimization-based method for the visualization of road networks.

For many way-finding and navigation tasks, the network connectivity is the most important feature, whereas Euclidean distances in the plane are rather meaningless. Therefore, we model the problem of generating a variable-scale map of a road network as a graph drawing problem – the graph that we aim to draw contains an edge for each road segment. We impose constraints on the graph layout to ensure, for example, that a user-selected focus region is displayed in a way similar to a large-scale geographic map and that the output map fits into a given map frame. Our objective is to minimize distortions that we measure at the edges of the input graph. As a result, distortions happen in regions where the graph is sparse, ideally in empty regions.

Our paper is structured as follows. We first give an overview on related work and the optimization technique we apply (Sect. 2). Then we present our new method for drawing road networks with focus regions in detail (Sect. 3) and we discuss our experiments (Sect. 4). Finally, we conclude the paper and give an outlook on future research (Sect. 5).

2 Related Work

In this section we discuss related work. We review fish-eye views in Sect. 2.1 and give an overview on approaches to map schematization in Sect. 2.2. In Sect. 2.3 we review optimization techniques for focus+context visualization, graph drawing, and cartographic displacement that, from an algorithmic point of view, are related to our method. Finally, in Sect. 2.4, we give a quick introduction into convex quadratic programming.

2.1 Fish-eye views

Fish-eye views have been frequently applied to city maps. Since streets and buildings are dense in the city center, the city center is mapped at a larger scale than suburbs. In Germany, such maps have been commercially produced at least since 1945, when Falk-Verlag published its first city map of Hamburg [20]. In recent years, fish-eye views have been frequently proposed for small-display cartography. Their application in dynamic applications, for example, car navigation [19], is promising. Normally, a large scale factor is defined for a focus point or focus region. The displacement of points outside the focus depends on the mapping function applied. The existing functions can be classified into two types: Mapping functions of the fist type distort the whole map outside the focus [5, 9], see Fig. 2 (left). In contrast, mapping functions of the second type only distort a certain region around the focus [8, 32], see Fig. 2 (right).

Yamamoto et al. [32] use the term *glue* for the distorted region. The points in this glue region are displaced away from the focus region by a certain distance, which results from a Bézier interpolation function. The part of the map that does not belong to the focus region or glue region is termed *context*. In this region the map is kept unchanged.

Polyfocal mapping functions allow a user to define multiple focus regions [13]. Leung and Apperley [18] compare geometric properties of different mapping fuctions. The application of fish-eye views for graph visualization is discussed by Sarkar and Brown [26]. Gansner et. al [6] have introduced a method that not only distorts an input graph geometrically but also collapses nodes in some parts of the graph to obtain a coarser representation.

A general problem with the existing mapping functions is that they introduce large distortions, especially when a large zoom factor is applied to the focus region. This motivates us to develop an optimization approach that allows constraints and objectives to be modeled. Our objective is to keep distortions small where they can be misleading, that is, at edges of the road network.

2.2 Map schematization

Map schematization does not necessarily need a mapping function. Instead, the positions of map objects on the plane can be determined with some method of map manipulation, for example, by optimization.

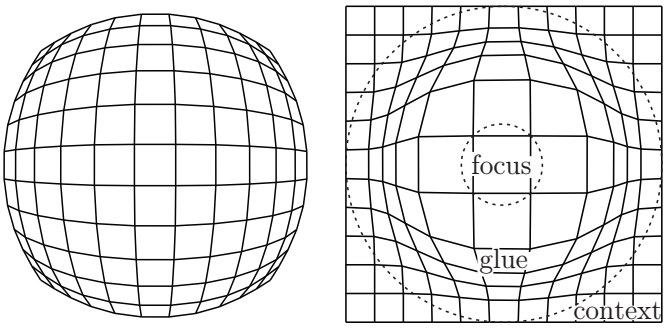


Fig. 2. Different mapping functions applied to the vertices of a regular grid; left: with the function by Harrie et al. [9] the scale decreases with increasing distance from the center; right: with the function by Yamamoto et al. [32] the input map is kept unchanged in the context region, i.e., outside an influence area (the glue region) of the enlarged focus region.

The term schematization is sometimes used synonymously with angular schematization, which means that each edge of a graph is drawn in a direction from a limited set of slopes. Angular schematization is frequently applied to metro maps, which allow users, for example, to efficiently decide where to change trains. Some algorithms exist for generating metro maps [11, 24, 30]. Research has been done to continuously transform a metro map into a geographic map [2] and to visualize its distortions [12].

According to Klippel et al. [16], schematization means more generally to "intentionally emphasize certain aspects of knowledge beyond technical necessity". Often schematization implies that the reference to geographic coordinates is lost. In a route map, for example, those parts of an input route that do not require navigation activities can be drastically scaled down. Route maps have been automatically generated by simulated annealing [1], which is a heuristic optimization technique. When deciding which parts of a route to scale down, a user's prior knowledge should be taken into account [27]. While route maps are in essence one-dimensional (they visualize a single line string plus some context information), destination maps visualize multiple roads yielding to a certain destination – such maps can also be produced with heuristic optimization methods [17].

Using a good geometric layout, certain parts of a network can be emphasized. Additionally, map generalization and coloring of certain map objects can be applied to put emphasis on a focus region [25, 33].

2.3 Related optimization approaches

Wang et al. [31] have presented a distortion minimization method for focus+context visualization of surface models. This method, however, has not been designed for network data and so the problem of edge crossings has not been addressed.

Geographic networks are sometimes drawn such that edge lengths represent travel times. This can be done by multidimensional scaling [29, 14], which, similar to our method, minimizes a quadratic objective function [4]. Using the method of Shimizu and Inoue [29], which was designed for travel-time visualization, Merrick and Gudmundsson [21] have developed an optimization-based visualization method that enlarges dense areas in a geographic network. Given the layout of a graph as input, the algorithm first computes a measure of centrality for the vertices of the graph. From this measure, the algorithm derives edge lengths that are desired for the output map. Then, the graph is deformed such that the edge lengths get close to the desired lengths and the edge slopes get close to their original values.

In contrast to the method of Merrick and Gudmundsson, our method is not driven by desired edge lengths. Instead, we require a user-defined scale factor for the focus region – this, of course, implies that the lengths of edges in the focus region become fixed. For all other edges, however, lengths are a priori unconstrained. Nevertheless, if we apply a scale factor to an edge e, it makes sense to apply the same

scale factor to all edges that share a vertex with e since otherwise the output map will not be similar to the input. Our measure of distortion expresses the degree to which the output map violates this rule.

Cartographic displacement is the problem of translating map objects such as to satisfy a proximity constraint, that is, in the output map two objects should not be closer to each other than a certain distance. A relaxed version of the problem where violations of the proximity constraint are allowed but penalized in the objective function can be solved by least-squares adjustment [10, 28]. Similarly, our method allows for distortions in the output map but minimizes their square sum. Least-squares adjustment is a special case of convex quadratic programming. In the more general model that we apply, hard constraints can be expressed in terms of inequalities. This allows us, for example, to ensure that the output map fits into a given map frame. The existing methods for cartographic displacement in road networks often avoid edge crossings by triangulating the vertices of the input network; the triangle edges are treated in the same way as the edges of the input network (yet they are not displayed on the map). A triangulation-based approach, however, does not work in our case since we want to take advantage of empty map space. If we filled the empty space with triangle edges, the network would become too rigid.

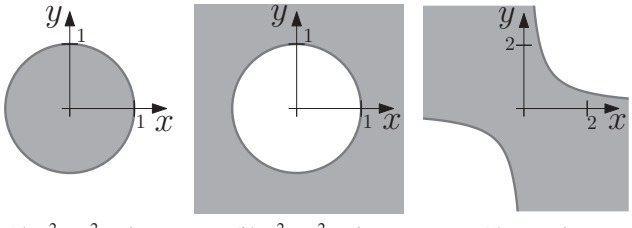
2.4 Convex quadratic programming

Mathematical programming is a common approach to optimization. Generally, a solution to the given problem is encoded in terms of variables. The optimization objective is expressed as a function in the variables; constraints in terms of equalities or inequalities are imposed on them. A lot of research has been done to solve certain classes of mathematical programs and there are sophisticated commercial solvers. If we choose a mathematical-programming approach, we directly profit by innovations that improve the general solvers.

Some classes of mathematical programs can be solved efficiently. For example, linear programs (LPs) can be efficiently solved, e.g., with Karmarkar's interior-point method [15]. An LP is a mathematical program with continuous (that is, real-valued) variables whose objective function and constraints contain linear terms only.

Quadratic programs (QPs) constitute a more general class of mathematical programs – their objective function can contain quadratic terms. Generally, a QP is given by an $m \times n$ matrix A, an $n \times n$ symmetric matrix D, an m-element vector b, and an n-element vector c. The problem is to find an n-element vector x of real numbers such that $cx + x^{\mathrm{T}}Dx$ is minimized and $Ax \geq b$ as well as $x \geq 0$ hold. In contrast to linear programming, the problem of solving quadratic programs is generally NP-hard [7], which means that we cannot hope to find an efficient solution for this class of problems. If the objective function is convex, however, the problem can be solved efficiently [23]. This is the case if the matrix D is positive semidefinite, that is, if $x^{\mathrm{T}}Dx \geq 0$ for every $x \in \mathbb{R}^n$. Such a QP is called a *convex quadratic program* (CQP). In a quadratically constrained quadratic program (QCQP) the constraints also contain quadratic terms. A QCQP can be solved efficiently if its objective function and constraints are convex [3]. Figure 3 visualizes a convex and two non-convex constraints.

Informally speaking, if we ensure that the objective function and the constraints are convex, we stay in the world of good-natured problems.



(a) $x^2 + y^2 \leq 1$ (b) $x^2 + y^2 \geq 1$ (c) $xy \leq 1$

Fig. 3. A convex set (a) and two non-convex sets ((b) and (c)) defined by quadratic inequalities. The points in the gray regions satisfy the inequalities.

3 Methodology

In this section we present our new method for drawing road networks with focus regions. We discuss the general idea behind our method in Sect. 3.1 and present a basic quadratic program in Sect. 3.2. Our experiments with the basic QP, which we present in Sect. 3.3, reveal that our method sometimes generates unwanted edge crossings. Therefore, we extend our QP in Sect. 3.4 to avoid edge crossings.

3.1 General ideas and preconditions

Given an input network, a user needs to select a focus region and to specify a zoom factor that she wishes to apply to this region.

- The *input network* is a graph $G = (V, E)$ where each node $u \in V$ has two coordinates X_u and Y_u in $\mathbb{R}$. Each edge $\{u, v\}$ in E corresponds to a road segment connecting u and v.
- The *focus region* is specified as a set $F \subseteq V$ of nodes. The aim is to scale this region up by a user-defined zoom factor $Z \geq 1$.

The problem is to enlarge the focus region while keeping the map within its original extent. Obviously, we can only satisfy this constraint by scaling down other parts of the map.

Our idea is to introduce *locally valid scale factors* that become determined through optimization. More precisely, for each node u in V, we introduce a scale factor s_u that is valid for the *star with center u*, that is, for the subgraph of G whose node set contains u plus all its neighbors and whose edge set contains an edge from u to each of its neighbors. In the output map, the star with center u should ideally be an exact copy of the original star with center u, scaled by factor s_u. It is impossible, however, to fulfill this rule for all stars. Therefore, we minimize a measure that expresses by how much this rule is violated.

In the following, we assume that the input graph G is connected – otherwise our method will draw the different connected components of G at arbitrary relative positions to each other. The restriction to connected graphs is tolerable when dealing with road networks. For example, if our map contains islands with roads, we may add ferry connections to the input graph in order to make the graph connected.

If two edges of the input graph cross each other (for example, via a bridge), we introduce the intersection point of both edges as a dummy node. At this node, we split each of the two edges involved into two new edges. When we compute the optimal layout of the network we treat the additional nodes in the same way as any other node. Thereby we account for the fact that bridges are important landmarks – they should not be lost when redrawing the network, and their position along a road should not change much.

3.2 Basic QP

We model a solution to our problem with three variables for each node $u \in V$, that is, the unknown coordinates $x_u, y_u \in \mathbb{R}$ and an unknown scale factor $s_u \in \mathbb{R}^+$. We now impose constraints on these variables.

First, we define a constraint to ensure that the map stays within its original extent, that is, within the smallest axis-parallel rectangle that contains the input map:

$$\begin{aligned} \min_{v \in V}\{X_v\} \leq x_u \leq \max_{v \in V}\{X_v\} \\ \min_{v \in V}\{Y_v\} \leq y_u \leq \max_{v \in V}\{Y_v\} \end{aligned} \qquad \text{for each } u \in V \tag{1}$$

Second, we fix the scale factor for each node in the focus region:

$$s_u = Z \qquad \text{for each } u \in F \tag{2}$$

For a node u not contained in the focus region, the scale factor is unknown to us – it will become determined through optimization, together with the coordinates of u in the output map. It remains to ensure that the scale factor s_u is valid for the neighborhood of u.

Suppose that we would express the idea of a locally valid scale factor with the constraint

$$\begin{aligned} s_u(X_v - X_u) = (x_v - x_u) \\ s_u(Y_v - Y_u) = (y_v - y_u) \end{aligned} \qquad \text{for each } u \in V, v \in \mathrm{Adj}(u), \tag{3}$$

where $\mathrm{Adj}(u)$ is the set of neighbors of u in G. With constraint (3), the star-shaped subgraph of G that contains u and its neighbors becomes scaled by the factor s_u. For two adjacent nodes i, j, however, we can only satisfy this constraint if we set $s_i = s_j$. Therefore, if G is connected, we have to select the same scale factor for all nodes in V. With constraint (3), it is thus impossible to design a *variable*-scale map.

In order to allow for different scale factors in different parts of the map, we introduce a relaxed version of constraint (3). We do not require that the neighborhood of node u is *exactly* mapped to scale. Instead, we allow for small distortions, which we measure based on residuals δx_{uv} and δy_{uv}. For this purpose, we introduce δx_{uv} and δy_{uv} as auxiliary variables into our model. The relaxed version of constraint (3) becomes

$$\begin{aligned} \delta x_{uv} &= s_u(X_v - X_u) - (x_v - x_u) \\ \delta y_{uv} &= s_u(Y_v - Y_u) - (y_v - y_u) \end{aligned} \quad \text{for each } u \in V, v \in \mathrm{Adj}(u). \tag{4}$$

If both u and v lie in the focus region F, we require

$$\delta x_{uv} = \delta y_{uv} = 0 \quad \text{for each } u, v \in F, v \in \mathrm{Adj}(u). \tag{5}$$

Thereby, we ensure that edges in the focus region indeed become enlarged by the zoom factor Z.

Our objective is to minimize the *weighted* square sum of residuals:

$$\text{Minimize} \sum_{u \in V} \sum_{v \in \mathrm{Adj}(u)} \left(\big(w(u,v) \cdot \delta x_{uv}\big)^2 + \big(w(u,v) \cdot \delta y_{uv}\big)^2 \right) \tag{6}$$

with $w(u,v) = 1/\sqrt{(X_v - X_u)^2 + (Y_v - Y_u)^2}$. With this weight setting we express that the validity of the scale factor s_u decreases with increasing distance from node u.

Our *basic QP* comprises constraints (1), (2), (4), and (5) plus the objective function (6). All constraints are linear and the objective function is convex since it doesn't contain mixed terms and all the weights are positive. Therefore, our basic QP can be solved efficiently.

3.3 Experiments with the basic QP

Figure 4 shows the solutions of our basic QP for three instances. In the first instance (Fig. 4(a)), the graph G is a grid of 12×12 nodes. Defining the four nodes in the center as the focus region and setting the zoom factor $Z = 2$, we obtain a deformed grid. The edge lengths of the square in the focus region increase by a factor of 2. When applied to a grid, the effect of our method is similar to that of the mapping function of Yamamoto et al. [32], which we reviewed in Sect. 2.1.

The second and third instances in Fig. 4 are similar to the first one, except that we removed some edges from the grid. In the second instance (Fig. 4(b)), the graph contains a bottleneck: It is possible to disconnect the lower left part (displayed in blue) from the rest of the graph by removing only two edges. If we scale the lower left part by a constant factor, residuals will appear only at the two connecting edges. Therefore, a solution that follows this strategy is optimal. The obtained map is very different from a map generated with a predefined map projection. In our graph-based optimization approach, the relative distances between two nodes of the graph are only relevant if the two nodes are connected by an edge. Since this conforms to the concept of a schematic road map, we consider the basic QP promising. The third instance (Fig. 4(c)), however, reveals a severe drawback of the basic QP: In the output map, different parts of the graph overlap each other. Therefore, we discuss how to forbid edge crossings in the next section.

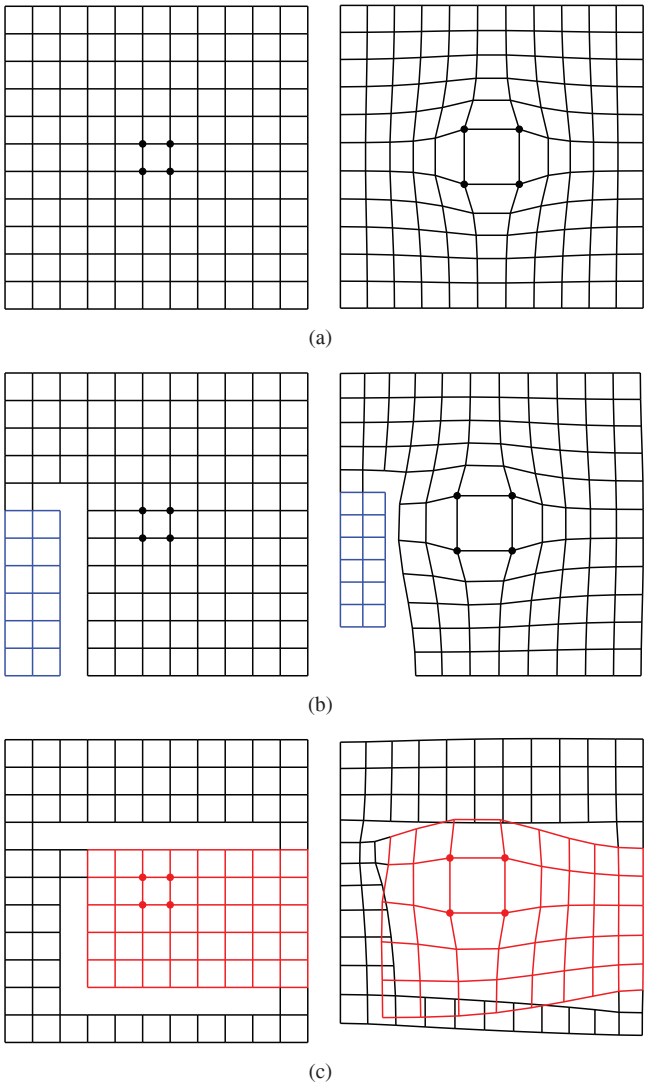


Fig. 4. Three instances processed with the basic QP. The figures display the input graph G before (left) and after transformation (right). The four nodes defining the focus region are displayed as dots. Subgraphs displayed in the same color correspond to each other.

3.4 Forbidding edge crossings

Our first attempt to avoid edge crossings was based on a constrained Delaunay triangulation of the map extent. More precisely, we partitioned the map extent into a set of non-overlapping triangles. We defined each vertex of G and each of the four corners of the map extent to be a vertex of the triangulation; each edge of G was constrained to become an edge of the triangulation. If we add all the triangle edges to G, edge crossings become unlikely. There are, however, two drawbacks of this approach. First, we cannot strictly guarantee a map without edge crossings and, second, our method does not make use of sparse and empty map regions, simply because the graph becomes (almost) evenly dense. The network becomes very inflexible.

Therefore, we tried a second triangulation-based approach. Instead of adding the triangle edges to the graph G, we introduced constraints to the basic QP that guarantee that each triangle of the triangulation keeps its orientation. Thereby, edge crossings were strictly avoided. Still, however, we lost significant flexibility. Furthermore, solving the QP with the additional constraints required much more time.

Our third and most successful approach to avoid edge crossings is not based on a triangulation. We now present this approach in detail.

Let $e = \{s,t\}$ and $f = \{u,v\}$ be two edges of G. We want to add a constraint to the basic QP that forbids both edges to cross. For this purpose, we define a directed line ℓ with the properties that, in the output map, both s and t lie to the left of or on ℓ and both u and v lie to the right of or on ℓ. The two edges e and f do not cross if and only if a line with these properties exists.

We can model the line ℓ with an unknown point (x,y) and an unknown direction vector whose components we denote by Δx and Δy, see Fig. 5. Introducing $x, y, \Delta x$ and Δy as variables, we can express the

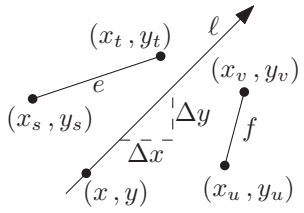


Fig. 5. By constraining the vertices of e to the left of line ℓ and the vertices of f to the right of ℓ, a crossing of e and f becomes impossible.

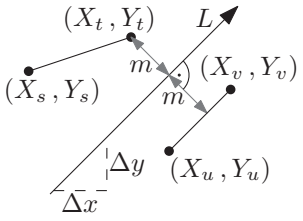


Fig. 6. The line L that maximizes the margin m for the edges $((X_s, Y_s), (X_t, Y_t))$ and $((X_u, Y_u), (X_v, Y_v))$ defines the slope of line ℓ.

properties required for ℓ in terms of the following inequalities.

$$s \text{ lies left of or on } \ell: \quad (y_s - y)\Delta x - (x_s - x)\Delta y \geq 0 \qquad (7)$$

$$t \text{ lies left of or on } \ell: \quad (y_t - y)\Delta x - (x_t - x)\Delta y \geq 0 \qquad (8)$$

$$u \text{ lies right of or on } \ell: \quad (y_u - y)\Delta x - (x_u - x)\Delta y \leq 0 \qquad (9)$$

$$v \text{ lies right of or on } \ell: \quad (y_v - y)\Delta x - (x_v - x)\Delta y \leq 0 \qquad (10)$$

Additionally, we need to forbid the trivial solutions $\Delta x = \Delta y = 0$. This could be done, for example, by setting $\Delta x = 1$.

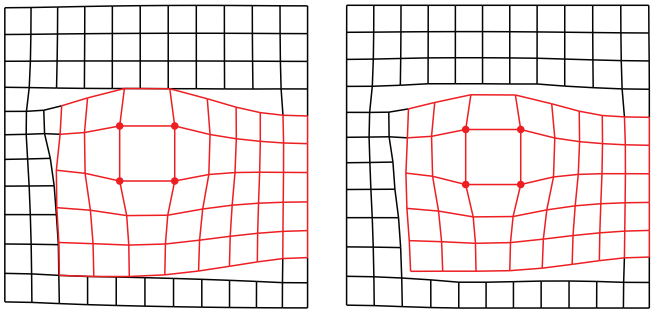
Unfortunately, the feasible region defined by constraints (7)–(10) is not convex. To see why, we analyze constraint (7). We substitute $(x_s - x)$ with a, Δy with b, and $(y_s - y)\Delta x$ with c. We obtain $ab \leq c$ and, for $c = 1$, we obtain $ab \leq 1$. This is a non-convex constraint, as we have seen in Fig. 3(c). Therefore, we cannot apply constraints (7)–(10) in a convex optimization framework. Nevertheless, we can add constraints to the QP that are even stricter. Thereby, we can avoid unwanted edge crossings.

We keep the solution set convex by fixing the slope of line ℓ, that is, we fix the unknowns Δx and Δy. Actually, with this change constraints (7)–(10) become linear.

When fixing Δx and Δy we need to be careful not to constrain the problem too much, that is, the road network should still have enough flexibility. In order to choose Δx and Δy, we construct the line L that, in the input map, separates the edges e and f with a maximum margin. In other words, we maximize the minimum distance between L and the edges e and f while requiring that the two edges lie on different sides of L. Then, we set the slope of ℓ to that of L. Figure 6 illustrates this idea. By maximizing the margin we remain relatively free in moving the four edge vertices: If we move a vertex, it is relatively unlikely that we will hit the line ℓ since the distance of the vertex to ℓ is relatively large. Note that finding a maximum-margin line is a standard problem that is well known in the context of support vector machines [22].

When setting up the QP, we could simply add constraints (7)–(10) for each pair of edges in G. Thereby, we would forbid all edge crossings in advance. This approach, however, has two drawbacks. First, solving the QP would take a long time since the size of the QP would become large: Both the number of variables and the number of constraints would become quadratic in the number of edges of G. Second, since with a fixed slope for ℓ constraints (7)–(10) are stricter than needed to avoid edge crossings, we would lose more flexibility than needed. Due to both reasons, we apply an optimistic strategy to handle the constraints: We first solve the basic QP without constraints (7)–(10). If we obtain a solution without edge crossings, we are done. Otherwise, we establish constraints (7)–(10) for each crossing. Then, we resolve the QP with the additional constraints. We iterate this process until we terminate with a solution without any edge crossing.

Figure 7(a) shows the result for the instance in Fig. 4(c) using our method to avoid edge crossings. In the example, a feasible solution was found after four iterations; constraints were added to forbid 25 crossings. Though the result does not contain edge crossings, we observe that some edges get extremely close to each other – sometimes a vertex of an edge even moves onto another edge. In order to avoid such cases, we introduce a minimal allowed distance ε. We change the right-hand sides of constraints (7)–(8) to $\varepsilon/2$ and the the right-hand sides of constraints (9)–(10) to $-\varepsilon/2$. Furthermore, we normalize the direction vector defined by Δx and Δy. With this modification, we avoid that the two edges handled by our constraints get closer than ε.



(a) $\varepsilon = 0.0$ (b) $\varepsilon = 0.4$ units

Fig. 7. Results of our method for the instance in Fig. 4(c). Edge crossings are avoided with constraints (7)–(10). The minimal allowed distance ε was set as specified – one unit equals the spacing of the grid.

4 Experimental Results

We implemented our method in Java using the Java library of the commercial optimizer ILOG CPLEX 12.1 to solve our quadratic program. In this section, we present results obtained with this implementation. In Sect. 4.1 we compare our method with the method of Yamamoto et al. [32], which is based on a predefined mapping function. In Sect. 4.2 we present statistics on the running time of our method.

4.1 A comparison with a predefined mapping function

We used the road network of central Boston in Fig. 1 (left) for our experiments. This data is freely provided online[1] by Massachusetts' Office of Geographic Information (MassGIS). We set the focus onto a highway interchange where multiple lanes occlude each other. In the output map, these lanes become visible, see Fig. 1 (right).

We tested the mapping function of Yamamoto et al. [32] on the same instance. In order to generate the fish-eye view, we have to define the glue region, that is, the region containing all distortions. We define the outer boundary of this region as a circle having the same center as the focus region. A large value for the radius r of this circle implies that there is relatively much space for distributing the distortions. On the other hand, the glue region has to lie within the original map frame, since otherwise the output map will exceed the bounds of that frame. For our experiment, we set $r = 640$ m, meaning that the glue region nearly touches the right boundary of the map frame.

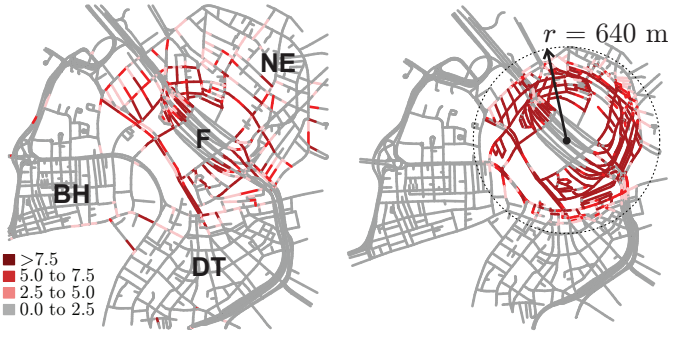
The result that we obtained with the predefined mapping function is shown in Fig. 8. In the focus region, this map does not differ from our result in Fig. 1 (right). Outside the focus region, however, the two maps are very different. Compared to our optimization method, the predefined mapping function highly distorts the map.

To measure the difference between the two output maps, we applied our optimization method a second time. This time, however, we fixed the nodes at their positions in the fish-eye view. In other words, we set, for each node u, the variables x_u and y_u in our QP to those coordinates that node u has in the map produced with the predefined mapping function. Consequently, we obtained the same output map as in Fig. 8. Still, the solver yielded a scale factor for each node and the residuals that are optimal under the fixed coordinates. The cost of this solution, that is, the value of objective (6) was 317.58. In comparison, our method yielded a cost of 80.65, that is, a reduction by 75 percent.

To find out where the two different methods distort the map, we visualize the residuals in Fig. 9. In the fish-eye view (Fig. 9(b)), all distortions happen between the focus region and the context region, that is, in the glue region. In radial direction from the center of the map,

[1] http://www.mass.gov/mgis/eotroads.htm

Fig. 8. The mapping function by Yamamoto et al. [32] applied to the map in Fig. 1. See Fig. 9(b) for the radius r defining the context region.



(a) residuals for the result in Fig. 1 (b) residuals for the result in Fig. 8

Fig. 9. Residuals at edges of the road network. For each edge $e = \{u,v\} \in E$ we show two line segments, one between u and the midpoint m of e and one between v and m. The color of the first segment reflects the weighted square sum of the residuals δx_{uv} and δy_{uv}; the color of the second segment reflects the weighted square sum of the residuals δx_{vu} and δy_{vu}. Note that these numbers are dimensionless.

the distortions first increase until they reach their maximum approximately at half distance between the focus and the context region. From there on, the distortions decrease. In the optimal layout (Fig. 9(a)), the distortions happen between certain areas. Basically, the road network of Boston contains three dense neighborhoods, namely Boston's North End (NE), Beacon Hill (BH), and Downtown (DT). Within Beacon Hill and Downtown the distortions are very small. Even in North End, which lies very close to the focus region, the distortions are relatively small. Most distortion takes place at the boundary of the focus region (F). Furthermore, some distortion takes place at links connecting the three neighborhoods and at the edge of the map. This result shows that our graph-based approach indeed avoids distortions in dense regions. Thereby, the characteristic shapes of Boston's neighborhoods are preserved. We repeated this experiments ten times with different focus regions spread over the whole map. On average, our method reduced costs by 81 percent compared to the predefined mapping function.

Fig. 10. Our result for the instance in Fig. 1 when fixing the positions of all vertices whose distance from the focus center exceeds 640 m.

The mapping function by Yamamoto et al. [32] ensures that vertices in the context region keep their position. This makes the method very efficient since only the focus region and the glue region need to be redrawn on a user request. Generally, our method allows all vertices to move, but we can use the same idea to speed up the processing, that is, we keep the positions of vertices in the context region fixed. Figure 10 shows the result of this approach applied to our Boston instance, using the same focus region and context region as in Fig. 8. Compared to our result in Fig. 1 the additional constraint caused an increase in cost by 38 percent. Still, the cost is 65 percent lower than with the predefined mapping function.

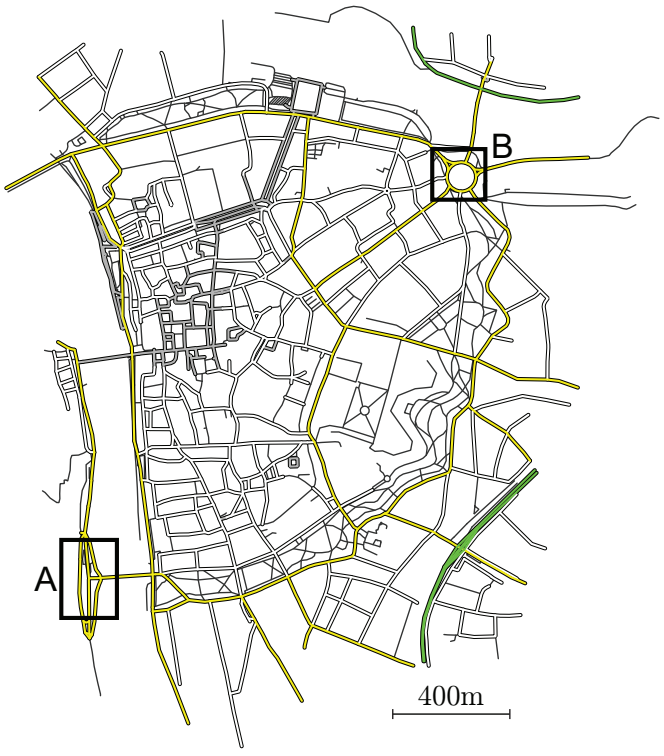
For a second test of our method we used road data for Würzburg, Germany from the OpenStreetMap[2] project. Figure 11 shows some results of these tests. The results support the statement that our method avoids distortions in dense regions. The figure also shows that our method allows us to define multiple disjoint focus regions.

4.2 Running time

We tested the running time of our method for instances of different sizes. Table 1 summarizes our results: Boston refers to the instance of Boston displayed in Fig. 1 (left); Würzburg2 refers to the network in Fig. 11(a); Würzburg1 refers to a subset of Würzburg2; Würzburg3 refers to a superset of Würzburg2. We solved all Würzburg instances with $Z = 3$ and focus region B in Fig. 11(a). For each instance, we list the number of nodes and the number of edges of the input graph, the number of iterations that we needed to find a solution without edge crossings, the total number of edge crossings that we needed to avoid by adding constraints (7)–(10), and the overall solution time measured in seconds CPU time. All experiments were performed on a Windows PC with 3 GB RAM and a 3.00 GHz Intel dual-core CPU.

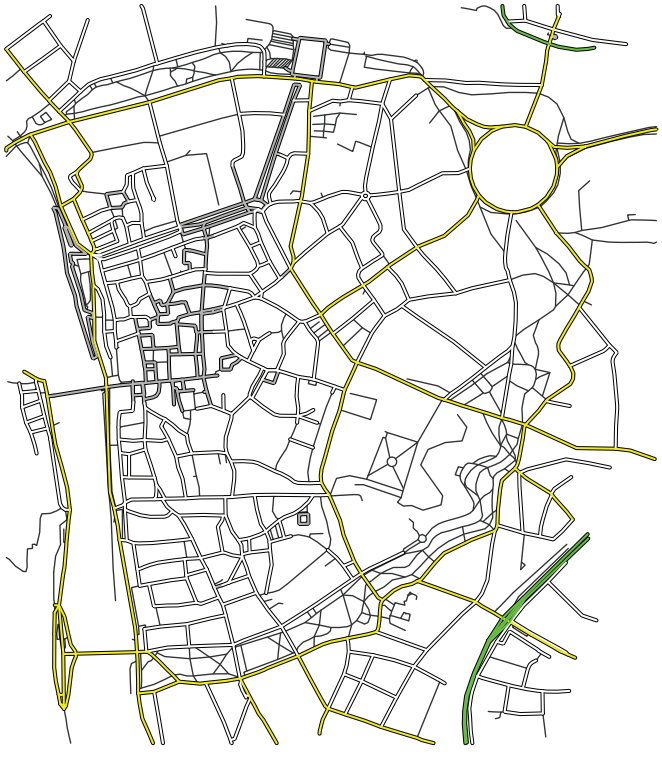
The statistics in Table 1 show that our method is fast enough to process instances of considerable size within a few seconds. For example, the solution of the instance Würzburg2, that is, the solution in Fig. 11(c) was found in about seven seconds – we think that instances of this size are typical for cartographic visualization on small displays. Even for the much larger instance Würzburg3, an optimal solution was found in modest time, that is, in about one minute.

[2] http://www.openstreetmap.org



(a) input network with two focus regions A and B

(b) result of applying $Z = 3$ to focus region A



(c) result of applying $Z = 3$ to focus region B

(d) result of applying $Z = 3$ to both focus regions A and B

Fig. 11. Results of our method for the road network of the center of Würzburg. The zoom factor $Z = 3$ was applied to two disjoint focus regions. Both focus regions contain complex traffic junctions, which our method enlarges. This can be used for navigation tasks where junctions are crucial decision points. The historical center of Würzburg has a dense network of pedestrian streets (gray-shaded lines). In this dense region, the distortions resulting from our method are very small.

instance	Boston	Würzburg1	Würzburg2	Würzburg3
# nodes	2852	465	2511	15941
# edges	3379	513	2965	18158
# iterations	3	4	4	5
# crossings	66	26	35	119
running time	6.70s	1.48s	7.08s	61.70s

Table 1. Results of our experiments on a Windows PC with 3 GB RAM and a 3.00 GHz Intel dual-core CPU; s stands for seconds CPU time.

For all instances, the solution obtained after the first iteration contained multiple edge crossings. Our idea was to forbid edge crossings only when they are encountered in a solution – this approach was successful since we needed to compute only a few (up to five) iterations and to forbid only a few (up to 119) edge crossings.

The result in Fig. 10, where all vertices in the context region were forced to keep their position, was obtained after 3.09 seconds. This is a reduction by 54 percent compared to the setting where all vertices were allowed to move. Keeping the context region unchanged is a good strategy if we have to deal with very large datasets, since it allows us to assume that our problem instances have constant size. Still, our method is much slower than the method of Yamamoto et al. [32], which solved the instance Würzburg3 in 0.02 seconds.

5 Conclusion and Future Work

We have presented a new method for enlarging a user-defined focus region in a road network, which is relevant for cartographic visualization. Our method ensures that with the enlarged focus region the map still fits into its original frame. To accomplish this task, some parts of the network are scaled down, which leads to map distortions. Our method minimizes a measure that quantifies distortions at road segments. This approach has considerable advantages compared to classical fish-eye views, which typically introduce large distortions to the network. We compared the result of our method with an existing fish-eye view technique. With our method, distortions at road segments are visibly smaller and, according to our measure of distortion, distortions are reduced by 75 percent. We also tested a variant of our method that deforms only a certain area around the focus region. This reduced the running time by 54 percent. Still, the distortions were 65 percent lower than in the classical fish-eye view.

Our method solves instances of about 3000 road segments in a few seconds. This may be fast enough to provide users with static maps. For real-time applications, however, we aim at speed-up techniques, which we will test in realistic settings. We also plan to look at a scenario where the focus region continuously moves over the map plane as the user drives on the road network. Furthermore, we plan to automatically select the roads relevant to a user and to integrate the problem of selecting roads and computing the layout of the network.

References

[1] M. Agrawala and C. Stolte. Rendering effective route maps: Improving usability through generalization. In *Proc. 28th Annu. Conf. Comput. Graphics (SIGGRAPH'01)*, pages 241–249, 2001.

[2] J. Böttger, U. Brandes, O. Deussen, and H. Ziezold. Map warping for the annotation of metro maps. *IEEE Comput. Graph.*, 28(5):56–65, 2008.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004.

[4] M. A. A. Cox and T. F. Cox. Multidimensional scaling. In C. Chen, W. Härdle, and A. Unwin, editors, *Handbook of Data Visualization*, pages 315–347. Springer, Berlin, Germany, 2008.

[5] D. Fairbairn and G. Taylor. Developing a variable-scale map projection for urban areas. *Comput. Geosci.*, 21(9):1053–1064, 1995.

[6] E. R. Gansner, Y. Koren, and S. C. North. Topological fisheye views for visualizing large graphs. *IEEE T. Vis. Comput. Gr.*, 11(4):457–468, 2005.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[8] F. Guerra and C. Boutoura. An electronic lens on digital tourist city-maps. In *Proc. 20th Int. Cartogr. Association Conf.*, pages 1151–1157, 2001.

[9] L. Harrie, L. T. Sarjakoski, and L. Lehto. A mapping function for variable-scale maps in small-display cartography. *J. Geospatial Eng.*, 4(2):111–123, 2002.

[10] L. Harrie and T. Sarjakoski. Simultaneous graphic generalization of vector data sets. *GeoInformatica*, 6(3):233–261, 2002.

[11] S.-H. Hong, D. Merrick, and H. A. D. do Nascimento. The metro map layout problem. In *Proc. Australasian Symp. Inform. Visualization - Volume 35*, (APVis '04), pages 91–100. Australian Computer Society, 2004.

[12] B. Jenny. Geometric distortion of schematic network maps. *Bulletin of the Society of Cartographers*, 40:15–18, 2006.

[13] N. Kadmon and E. Shlomi. A polyfocal projection for statistical surfaces. *Cartogr. J.*, 15(1):36–41, 1978.

[14] C. Kaiser, F. Walsh, C. J. Q. Farmer, and A. Pozdnoukhov. User-centric time-distance representation of road networks. In *Proc. 6th Int. Conf. Geographic Inform. Sci.*, (GIScience'10), pages 85–99. Springer, Berlin, Germany, 2010.

[15] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proc. 16th Annu. ACM Symp. Theory Comput.*, (STOC '84), pages 302–311. ACM, 1984.

[16] A. Klippel, K.-F. Richter, T. Barkowsky, and C. Freksa. The cognitive reality of schematic maps. In *Map-based Mobile Services – Theories, Methods and Implementations*, pages 57–74. Springer, Berlin, Germany, 2005.

[17] J. Kopf, M. Agrawala, D. Bargeron, D. Salesin, and M. Cohen. Automatic generation of destination maps. *ACM T. Graphic.*, 29(6):158:1–158:12, 2010.

[18] Y. K. Leung and M. D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM T. Comput.-Hum. Int.*, 1:126–160, June 1994.

[19] Q. Li. Variable-scale representation of road networks on small mobile devices. *Comput. Geosci.*, 35(11):2185–2190, 2009.

[20] H. Lintzöft. 50 Jahre Falkplan. In *Patent-Plan Hamburg (Reprint anlässlich 50 Jahre Falkplan 1945-1995)*. Falk-Verlag, Hamburg, Germany, 1995.

[21] D. Merrick and J. Gudmundsson. Increasing the readability of graph drawings with centrality-based scaling. In *Proc. 2006 Asia-Pacific Symp. Inform. Visualization - Volume 60*, APVis '06, pages 67–76. Australian Computer Society, 2006.

[22] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE T. Neural Networ.*, 12(2), 2001.

[23] K. G. Murty. *Linear complementarity, linear and nonlinear programming*. Heldermann, Berlin, 1988.

[24] M. Nöllenburg and A. Wolff. Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE T. Vis. Comput. Gr.*, 17(5):626–641, 2011.

[25] K.-F. Richter, D. Peters, G. Kuhnmünch, and F. Schmid. What do focus maps focus on? In *Proc. Int. Conf. Spatial Cognition VI: Learning, Reasoning, and Talking about Space*, volume 5248 of *Lecture Notes In Artificial Intelligence*, pages 154–170. Springer, Berlin, Germany, 2008.

[26] M. Sarkar and M. H. Brown. Graphical fisheye views. *Commun. ACM*, 37:73–83, December 1994.

[27] F. Schmid. Knowledge-based wayfinding maps for small display cartography. *J. Location Based Services*, 2(1):51–83, 2008.

[28] M. Sester. Optimization approaches for generalization and data abstraction. *Int. J. Geogr. Inf. Sci.*, 19(8–9):871–897, 2005.

[29] E. Shimizu and R. Inoue. Time-distance mapping: visualization of transportation level of service. In *Proc. Symp. on Environmental Issues Related to Infrastructure Development*, pages 221–230, 2003.

[30] J. Stott, P. Rodgers, J. C. Martinez-Ovando, and S. G. Walker. Automatic metro map layout using multicriteria optimization. *IEEE T. Vis. Comput. Gr.*, 17:101–114, January 2011.

[31] Y.-S. Wang, T.-Y. Lee, and C.-L. Tai. Focus+context visualization with distortion minimization. *IEEE T. Vis. Comput. Gr.*, 14(6):1731–1738, 2008.

[32] D. Yamamoto, S. Ozeki, and N. Takahashi. Focus+glue+context: an improved fisheye approach for web map services. In *Proc. 17th Annu. ACM Symp. Advances Geogr. Inform. Syst. (ACM-GIS'09)*, pages 101–110. ACM, 2009.

[33] A. Zipf and K.-F. Richter. Using focus maps to ease map reading: Developing smart applications for mobile devices. *Künstliche Intelligenz (KI)*, 02(4):35–37, 2002.