An Algorithmic Framework for Labeling Road Maps

Benjamin Niedermann[1(✉)] and Martin Nöllenburg[2]

[1] Karlsruhe Institute of Technology, Karlsruhe, Germany
benjamin.niedermann@kit.edu
[2] TU Wien, Vienna, Austria

Abstract. Given an unlabeled road map, we consider, from an algorithmic perspective, the cartographic problem of placing non-overlapping road labels embedded in the roads. We first decompose the road network into logically coherent road sections, i.e., parts of roads between two junctions. Based on this decomposition, we present and implement a new and versatile framework for placing labels in road maps such that the number of labeled road sections is maximized. In an experimental evaluation with road maps of 11 major cities we show that our proposed labeling algorithm is both fast in practice and that it reaches near-optimal solution quality, where optimal solutions are obtained by mixed-integer linear programming. In direct comparison, our algorithm consistently outperforms the standard OpenStreetMap renderer Mapnik.

1 Introduction

Due to the increasing amount of geographic data and its continual change, automatic approaches become more and more important in cartography. This particularly applies to the time-consuming and demanding task of label placement and much research has been done on its automation. Badly placed labels of features of interest can easily make maps unreadable [4]. Depending on the type of map feature, label placement is done differently. For *point features* (e.g., cities on small-scale maps) labels are typically placed closely to that feature, while for *line features* (e.g., roads, rivers) the name is either placed along or inside the feature. The latter approach is also used for *area features* (e.g., lakes). Regardless of the applied technique and feature type, labels should not overlap each other and clearly identify the features [8].

The cartographic label placement problem has also attracted the interest of researchers in computational geometry and has been thoroughly investigated from both the practical and theoretical perspective [13, Chapter 58.3.1], [14]. While algorithms for labeling point features get a lot of attention, much less work has been done on line features and area features. In this paper we address labeling line features, namely labeling the entire road network of a road map. We take an algorithmic, mathematical perspective on the underlying optimization problem and build on our recent theoretical results for labeling tree-shaped networks [3]. We apply the quality criteria for label placement in road maps elaborated by

J.A. Miller et al. (Eds.): GIScience 2016, LNCS 9927, pp. 308–322, 2016.
DOI: 10.1007/978-3-319-45738-3_20

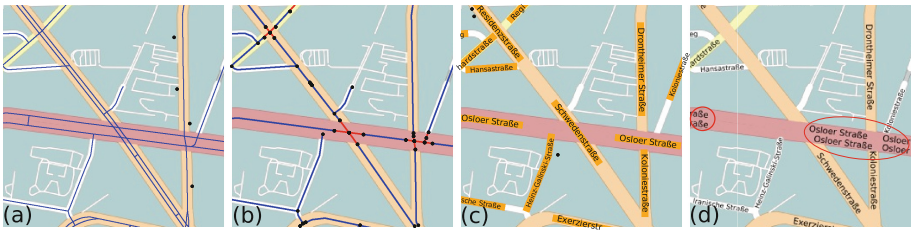

Fig. 1. The presented workflow. (a) The road network given by polylines (thin, blue segments). (b) Phase 1: a graph G is created whose embedding is the simplified road network; blue segments: road sections, red segments: junction edges. (c) Phase 2: creating the labeling using G. (d) A labeling produced by the OSM renderer Mapnik. The six labels of road *Osloer Straße* are enclosed by red ellipses. (Color figure online)

Chirié [2] based on interviews with cartographers. They include that (C1) labels are placed inside and parallel to the road shapes, (C2) every road section between two junctions should be clearly identified, and (C3) no two road labels may intersect. Similar criteria have been described in a classical paper by Imhof [4].

Variations of embedded labels have been considered in road maps before. Chirié [2] and Strijk et al. [11, Chapter 9] presented simple, local heuristics that place non-overlapping labels based on a discrete set of candidate positions – in contrast we consider the problem globally applying a continuous sliding model. Seibert and Unger [10] utilized the geometric properties of grid-based road networks and proved that it is NP-complete to decide whether at least one label can be placed for each road. For the same grid-based setting Neyer and Wagner [6] evaluated a practically efficient algorithm that is not applicable for general road networks.

Road labeling with embedded labels has also been considered for interactive and dynamic maps. Maass and Döllner [5] provided a heuristic for labeling interactive 3D road maps taking obstacles into account. Vaaraniemi et al. [12] presented a study on a force-based labeling algorithm for dynamic maps considering both point and line features. Schwartges et al. [9] investigated embedded labels in interactive maps allowing panning, zooming and rotation of the map. They evaluated a simple heuristic for maximizing the number of placed labels.

For labeling point features a typical objective is to maximize the number of non-overlapping placed labels, because every placed label enhances the map with further information. While this is mostly true for point features, maximizing the number of labels is not the right objective for label placement of roads since not every label that is placed necessarily contributes more information to the map. For example, consider the placed labels of the road *Osloer Straße* in Fig. 1(d). We can easily remove some of those labels without losing any information, because the map user can still identify the same road sections; see Fig. 1(c). In online map services, however, one often finds such redundant labels; see the full version of this paper [7] for two examples. Some roads may have unnecessarily many labels, which ma y in turn cause others to remain completely unlabeled.

Hence, the user cannot identify such roads on the map, a real disadvantage if headed for that road. Due to these observations we do not aim to maximize the number of labels, but the number of labeled *road sections*. For the purpose of this paper, a *road section* forms a connected piece of the road network that logically belongs together, e.g., a part of a road between two junctions or a part that distinguished by its color or width. Our algorithm, however, is independent of the actual definition of road sections; any partition of the road network into disjoint road sections can be handled. We say that a road section is *labeled* if a label (partly) covers it.

As the underlying model for maximizing labeled road sections we re-use the planar graph model that has been introduced in our theoretical companion paper [3]. In that paper we proved that labeling a maximum number of road sections is NP-hard, even for planar graphs and if no road consists of multiple branches. However, we presented a polynomial-time algorithm for the case that the road graph is a tree. While this result for trees is mostly of theoretic interest (road networks rarely form trees), we will show in this paper that our tree-based algorithm can be used successfully as the core of an efficient and practical road labeling algorithm that produces near-optimal solutions.

Contribution and Outline. We introduce a versatile algorithmic framework for placing non-overlapping labels in road networks maximizing the number of labeled road sections. We keep the algorithmic components easily exchangeable. In Sect. 2 we discuss and expand the model introduced in [3]. Afterwards, we present a workflow for labeling road networks in two phases; see Fig. 1.

Phase 1 (Sect. 3). We translate the given road network into a semantic representation (an abstract road graph) that identifies pieces of the road network that belong semantically together. To that end, we simplify the road network, e.g., we merge lanes closely running in parallel. By design this simplification maintains the overall geometry of the road network and only merges structures in the data that should not be labeled independently. Phase 1 is not part of the labeling optimization process.

Phase 2 (Sect. 4). Based on the abstract road graph, we create an actual labeling using one of three algorithms: a naive base-line algorithm, a heuristic extending our tree-based algorithm [3] and a mixed-integer linear programming (MILP) formulation.

As proof of concept we implemented the core of the framework only taking the most important cartographic criteria into account. However, with some engineering it can be easily enhanced to more complex models, e.g., enforcing minimum distances between labels, abbreviating road names, or using alternative definitions of road sections. In Sect. 5 we present a detailed evaluation of our framework on 11 sample city maps. Due to its availability and popularity in practice, we compare our results against the standard OpenStreetMap (OSM) renderer Mapnik as a representative of local heuristics; it uses a strategy similar to [2,11]. We show that our tree-based algorithm is fast and yields near-optimal labelings that outperform Mapnik.

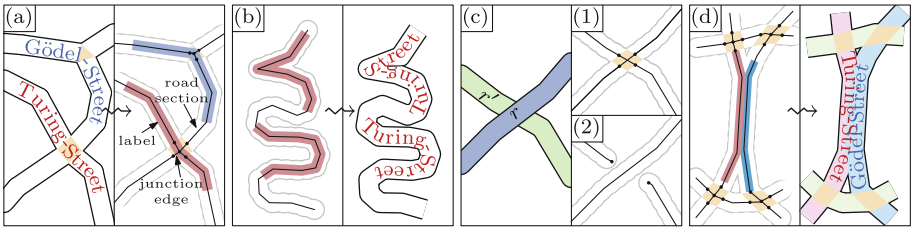

Fig. 2. Illustration of model and arising issues. (a) Sketch of a road network and its abstract road graph. (b) Labels are possibly curvy and have sharp bends making the text hardly legible. (c) Issue 2: two ways to represent bridges and tunnels in the abstract road graph. (d) Issue 4: the text representation of labels may overlap, while the curve representation in the abstract road graph does not. (Color figure online)

2 Semantic Representation of Road Networks

At any given scale, road networks are typically drawn as follows. Each road or road lane is represented as a thick, polygonal curve, i.e., a polygonal curve with non-zero width; see the background of Fig. 1(a). If two (or more) such curves intersect, they form junctions. If two or more lanes of the same road closely run in parallel they merge to one even thicker curve such that individual lanes become indistinguishable. We then want to place road labels inside those thick curves. More precisely, a *road label* can again be represented as a thick curve (the bounding shape of the road name) that is contained in and parallel to the thick curve representing its road; see Fig. 1(c).

For the purpose of this paper it is sufficient to use a simplified representation, which represents the road network and its labels as thin curves instead [3]. More precisely, a road network is modeled as a planar embedded *abstract road graph* whose edges correspond to the skeleton of the actual thick curves. In this model a label is again a thin curve of certain length that is contained in the skeleton. Following the cartographic quality criteria (C1)–(C3), we want to place labels, i.e., find sub-curves of the skeleton, such that (1) each label starts and ends on road sections, but not on junctions, (2) no two labels overlap, and (3) a maximum number of road sections are labeled. Requiring that labels end on road sections avoids ambiguous placement of labels in junctions where it is otherwise unclear how the road passes through it. Note that this does not forbid labels across junctions. From a labeling of the abstract road graph it is straight-forward to transform each label back into its *text representation* by placing the individual letters of each label along the thick curves; see Fig. 2(a).

Abstract Road Graph Model. We have introduced the abstract road graph in [3], but for the convenience of the reader we repeat it here, see also Fig. 1(b) and Fig. 2(a). A road network (in an abstract sense) is a planar geometric *graph* $G = (V, E)$, where each vertex $v \in V$ has a position in the plane and each edge $\{u, v\} \in E$ is represented by a polyline whose end points are u and v. Each edge further

has a *road name*. A maximal connected subgraph of G consisting of edges with the same name forms a *road* R. The length of the name of R is denoted by $\lambda(R)$. Each edge $e \in E$ is either a *road section*, i.e., the part of a road in between two junctions, or a *junction edge*, which models road junctions. Formally, a *junction* is a maximal connected subgraph of G that only consists of junction edges. Typically, when two roads cross, a junction is a star with one center vertex and four outer vertices, but more complex junctions are possible. We require that no two road sections in G are incident to the same vertex and that vertices incident to road sections have at most degree 2. Thus, the road graph G decomposes into road sections, separated by junctions.

We say a point p lies on G, if there is an edge $e \in E$ whose polyline contains p. Hence, a polyline ℓ (in particular a single line segment) lies on G if each point of ℓ lies on G. Further, ℓ *covers* e, if there is a point of ℓ that lies on e. If each point of e is covered by ℓ, e is *completely covered*. The *geodesic distance* of two points on G is the length of the shortest polyline on G connecting both points.

A *label* of a road R is a simple open polyline ℓ on G that has length $\text{len}(\ell) = \lambda(R)$, ends on road sections of G, and whose segments only lie on edges of R. The start point of ℓ is denoted as the *head* $h(\ell)$ and the endpoint as the *tail* $t(\ell)$. Obviously, the edges that are covered by ℓ form a path $\mathcal{P}_\ell = (e_1, e_2, \cdots, e_{k-1}, e_k)$ such that e_1, and e_k are (partly) covered and $e_2, \ldots, e_{k-1}$ are completely covered by ℓ. If e_i is a road section (and not a junction edge), we say that e_i is *labeled* by ℓ. We restrict ourselves to *well-shaped* labels, i.e., labels that are not too curvy or do not contain broken type setting due to sharp bends; see Fig. 2(b). Similar to Schwartges et al. [9], we apply a local criterion to decide whether a label is well-shaped; see also [7]. A *labeling* $\mathcal{L}$ for a road network is a set of mutually non-overlapping, well-shaped labels, where two labels ℓ and ℓ' *overlap* if they intersect in a point that is not their respective head or tail.

Following the criteria (C1)–(C3), the problem MaxLabeledRoads is to find a labeling $\mathcal{L}$ that labels a maximum number of road sections, i.e., no other labeling labels more road sections. In [3] we showed that MaxLabeledRoads is NP-hard in general, but can be solved in $O(|V|^3)$ time if G is a tree.

Shortcomings for Real-world Road Networks. While the abstract road graph model allows theoretical insights, we cannot directly apply it to real-world road networks. Due to the following issues, we need to invest some effort in a preprocessing phase (see Sect. 3) to guarantee that the resulting labels in the text representation do not overlap, look nice and are embedded in the roads' shapes.

Issue 1: If lanes run closely in parallel, their drawings in the road network merge to one thick curve and individual lanes become indistinguishable. Hence, in our abstract model, such lanes should be aggregated to a single road section that represents the skeleton of the merged curve, and labels should be contained in it; see Fig. 1(c).

Issue 2: Real-world road networks are not planar, but edges may cross, namely at tunnels and bridges; see Fig. 2(c). To avoid overlaps between labels placed on those road sections, we either can model the intersection as a regular junction of two roads or we split one into two shorter road sections that do not

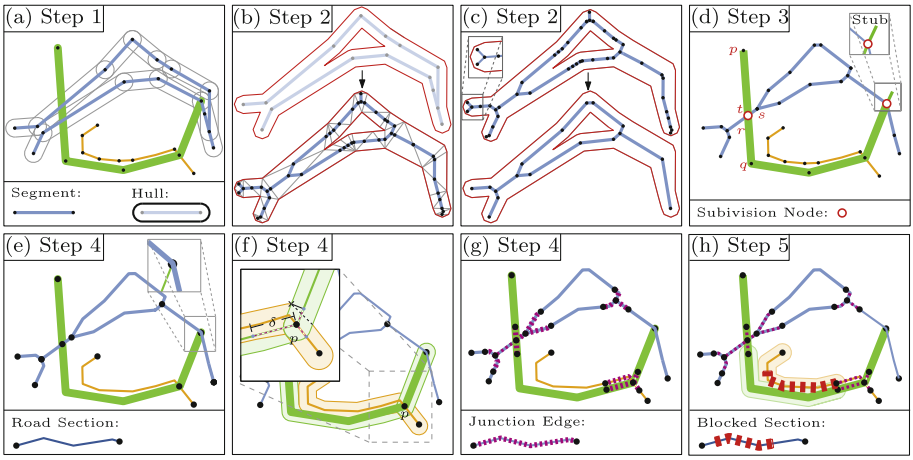

Fig. 3. Illustration of the steps applied in Phase 1. Segments of the same thickness and color have the same road name. For more details see the description of Phase 1. (Color figur online)

cross the other road section. In both cases the road graph becomes planar. For our prototype we use the first variant (also used by Mapnik), because more road sections can be labeled.

Issue 3: In real-world road networks some road sections are possibly so long that the label should be repeated after appropriate distances.

Issue 4: Labels have a certain font size so that when transforming an abstract label curve into its text representation, labels of different roads may overlap due to their road sections being too close; see Fig. 2(d).

3 Phase 1 – Construction of Abstract Road Graphs

The first phase of our framework consists of transforming the input road network data into an abstract road graph while resolving the four issues mentioned in Sect. 2. Typically, road networks are given as a set of polylines that describe the roads and road lanes. Individual polylines do not necessarily form semantic components such as road sections. So as a first step, we break all polylines down into individual line segments (whose union forms the road network). Let L be the set of all these line segments. We further require that each line segment $l \in L$ is annotated with its *road name* $\mathrm{rn}(l)$, the stroke width $\mathrm{st}(l)$ and the color $\mathrm{co}(l)$ that are used to draw l, and finally the *font size* $\mathrm{fs}(l)$ that shall be used to display the name. We say that two line segments $l, l' \in L$ are *equally represented* if $\mathrm{st}(l) = \mathrm{st}(l')$ and $\mathrm{co}(l) = \mathrm{co}(l')$. We assume that $\mathrm{fs}(l) < \mathrm{st}(l)$ for any l; otherwise we set $\mathrm{st}(l) := \mathrm{fs}(l)$.

The workflow consists of the following five steps; see Fig. 3. (1) Identification. Identify single *road components*, i.e., sets of line segments in the road

network data that have the same name, are equally represented, and form a connected component. (2) Simplification. Simplify each road component such that lanes running closely in parallel are aggregated. (3) Planarization. Replace bridges and tunnels by artificial junctions. (4) Transformation. Transform the segment representation into an abstract road graph. (5) Resolving Overlaps. Identify mutual overlaps of road sections and block them for label placement.

Below we describe each step in more detail. We define the *hull* of a line segment $l \in L$ to be the region of points whose Euclidean distance to l is at most $\mathrm{st}(l)$; see Fig. 3(a). The hull of a polyline is then the union of its segments' hulls. We approximate hulls by simple polygons.

Step 1 – Identification. For each road name n, each color c and each font size f we define the intersection graph of the hulls of the line segments $L_{n,c,f} = \{l \in L \mid \mathrm{rn}(l) = n,\ \mathrm{co}(l) = c \text{ and } \mathrm{fs}(l) = f\}$. In this intersection graph each hull is a vertex and two vertices are connected if and only if the corresponding hulls intersect. In each (non-empty) intersection graph we identify all connected components, which we call *road components*; e.g., in Fig. 3(a) the blue segments form a road component. Thus, based on L we obtain a set $\mathcal{C}$ of road components. By definition, each component $C \in \mathcal{C}$ has a unique name $\mathrm{rn}(C)$, stroke width $\mathrm{st}(C)$, color $\mathrm{co}(C)$ and font size $\mathrm{fs}(C)$.

Step 2 – Simplification. For each road component $C \in \mathcal{C}$ we geometrically form the union of the according hulls. Thus, the result is a simple polygon P (possibly with holes) describing the contour of the road component; see Fig. 3(b), top. Following Bader and Weibel [1] we use the conforming Delaunay Triangulation of P to construct a *skeleton* as a linear representation of C. In this way, after some further simplifications (see Fig. 3(c)), we obtain a skeleton for each component C such that labels centered on the skeleton are guaranteed to be contained in P. This resolves Issue 1. We annotate each skeleton edge with the name, stroke width, color and font size of C. For more details see [7].

Step 3 – Planarization. So far polylines describing the skeletons of different road components may intersect at other points than their end points, e.g., polylines representing bridges and tunnels may cross other polylines. As motivated in Sect. 2, we subdivide these polylines to resolve intersections; see Fig. 3(d). More precisely, if two line segments $\overline{pq}$ and $\overline{rs}$ of two polylines intersect at a point t, we replace them by the four segments $\overline{pt}$, $\overline{tq}$, $\overline{rt}$ and $\overline{ts}$. We do the intersection tests with a certain tolerance to identify T-crossings safely. However, this may yield short stubs that protrude junctions slightly; we remove those stubs. This resolves Issue 2 and yields a set of annotated polylines only intersecting in vertices.

Step 4 – Transformation. Next we create the abstract road graph from the polylines of the previous step. As a result of Step 3 we know that any two polylines intersect only in vertices. We first take the union of all polylines, identify vertices that are common to two or more polylines and mark these vertices as *junction seeds*. This induces already a planar graph $G = (V, E)$ with polyline edges whose vertices V are either junction seeds or have degree 1. It remains to partition the edges of G into road sections and junction edges. Initially, we mark all edges as road sections. We distinguish two types of junction seeds in G.

If a junction seed v has degree at least 3, only two of its incident edges e and e' belong to the same road R and all other incident edges belong to different roads (and have a different road type than R) then we do not create any junction edges at v, see Fig. 3(e), small box. Since R is the only road that may use the junction at v and it is visually clear that all other roads end at v we can safely treat v as an internal vertex of a road section of R. So we disconnect all incident edges of v except e and e' from v and let each of them end at its own slightly displaced copy of v. The edges e and e' are merged at v and the new edge remains a road section. This resolves the situation as desired.

For all other junction seeds we create junction edges as follows. Let v be a junction seed and let E_v be the set of edges incident to v. We intersect the hulls of all edges in E_v and project their intersection points onto the corresponding edges, see Fig. 3(f). For each edge $e \in E_v$ we determine the projection point p_e that is farthest away from v (in geodesic distance). If the distance between p_e and v exceeds a given threshold δ, we shift p_e to the point on e that has distance δ from v. Now we subdivide e at p_e and mark the edge $\{v, p_e\}$ as a junction edge; the other edge at p_e (if non-empty) remains a road section. The threshold δ ensures that roads running closely in parallel are not completely marked as junction edges. Figure 3(g) shows the resulting abstract road graph.

To resolve Issue 3 we subdivide road sections whose length exceeds a certain threshold (in our experiment 350 pixels) by inserting a very short junction edge.

Step 5 – Resolving Overlaps. By Step 2 the hulls of edges that belong to the same road component do not overlap. However, if two sections of different roads run closely in parallel, their hulls (and hence their labels) may overlap. We identify overlaps of the hulls of non-incident edges in G and block the corresponding parts of the edge whose road is less important for placing labels; ties are broken arbitrarily. More complex approaches using road displacement could be applied, however, we have chosen a simple solution. By design hulls of incident edges may only overlap if both are junction edges; those overlaps are handled by the labeling algorithms; see Sect. 4. This resolves Issue 4.

4 Phase 2 – Label Placement in Road Graphs

In this section we present the four different methods for solving MaxLabeledRoads that we subsequently evaluate in our experiments in Sect. 5. Furthermore, we describe a technique for decomposing road graphs into several smaller, independent components that may speed up computations.

4.1 Labeling Methods

BaseLine. An obvious base-line heuristic to obtain lower bounds is to simply place a well-shaped label on each individual road section that is long enough to admit such a label without extending into any junctions. We use this approach to show that it is beneficial to position labels across junctions.

Mapnik. Mapnik (http://mapnik.org) is a standard open source renderer for OpenStreetMap that includes an road labeling algorithm. The algorithm

iteratively labels so-called *ways*, which are polylines describing line features in OpenStreetMap. Along each way it places labels with a certain spacing and locally ensures that labels do not intersect already placed labels of other ways. It does not use any semantic structure from the road network (e.g., road sections), but relies on how the contributors of OpenStreetMap modeled single ways. We may run the rendering algorithm and extract all placed labels from its output.

Tree. The tree-based heuristic makes use of our recently proposed algorithm that optimally solves MaxLabeledRoads if G is a tree [3]. The basic idea for trees is that a placed label splits the tree into several independent sub-trees, which then are labeled recursively. Using dynamic programming we reuse already computed results so that the algorithm's complexity becomes polynomial, namely $O(|V|^5)$ running time and $O(|V|^2)$ space. Applying some further intricate modifications we improved this to $O(|V|^3)$ time and $O(|V|)$ space, and $O(|V|^2)$ time if each road in G is a path. We omit the details (see [3]) and use that algorithm as a *black box*. If G is a tree, our heuristic optimally labels G. Otherwise it computes a spanning tree T on G using Kruskal's algorithm and computes an optimal labeling for T. We construct T such that all road sections of G are contained in T. Since a road section is only incident to junction edges, this is always possible. In Sect. 5 we show that large parts of realistic road networks can actually be decomposed into paths and trees without losing optimality.

Milp. In order to provide upper bounds for the evaluation of our labeling algorithms, we implement a mixed-integer linear programming (MILP) model that solves MaxLabeledRoads optimally on arbitrary abstract road graphs. The basic idea is to discretize all possible label positions and to restrict the space of feasible solutions to non-overlapping sets of labels; see also [7]. Although solving a MILP is generally NP-hard, we can apply specialized solvers to find optimal solutions for reasonably sized instances in acceptable times.

4.2 Decomposition of Road Networks

We may speed up both our heuristic Tree and the exact approach Milp by decomposing the road graph into smaller, independent components to be labeled separately, i.e., components whose individual optimal solutions compose to a conflict-free optimal solution of the initial road graph. Such a decomposition allows us to compute solutions in parallel with either of the above methods and it further decreases the total combinatorial complexity. The decomposition rules guarantee that the labelings of the components can always be merged without creating any label overlaps. We name this technique D&C and sketch the decomposition and composition steps; see also [7].

Step 1 – Decomposition. For many road sections, e.g., long sections, of real-world road networks labels can be easily placed preserving the optimal labeling. We iterate through the edges of G and cut or remove some of them if one of the following rules applies. As a result the graph decomposes into independent connected components; see Fig. 4(a)–(d). Let e be the currently considered edge and let R be the road of e.

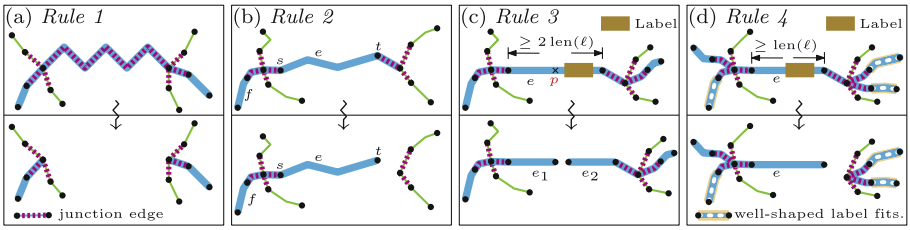

Fig. 4. Illustration of the rules. Edges of the same thickness and color belong to the same road. (Color figure online)

Rule 1. If e is a junction edge and it cannot be completely covered by a well-shaped label, i.e., e is not well-shaped, then remove e.

Rule 2. If e is a road section that ends at a junction that is not connected to any other road section of R, then detach e from that junction.

Rule 3. If e is a road section, a well-shaped label ℓ fits on e, and e is at least twice as long as ℓ, then cut e at its midpoint.

Rule 4. If e is a road section, a well-shaped label ℓ fits on e, and e is connected to a junction that is only connected to road sections of R that may completely contain a well-shaped label, then detach e from that junction.

On each edge we apply at most one rule. If we apply *Rule 3* or *Rule 4* on an edge e, we call e a *long-edge*. Afterwards, we determine all connected components of the remaining graph G', which are then independently labeled.

Step 2 – Label Placement. For the constructed components we compute solutions in parallel with either of the above methods.

Step 3 – Composition. Finally, we compose the labelings of the second step to one labeling. Due to the decomposition, no two labels of different components can overlap. If a long-edge e is not labeled, we place a label on it, which is possible by definition. We adapt the algorithms of Step 2 such that they do not count labeled road sections that were created by *Rule 3*, but we count the corresponding long-edge in this step.

5 Evaluation

We evaluate our framework and in particular the performance of our new tree-based labeling heuristic by conducting a set of experiments on the road networks of 11 North American and European cities; see Table 1. While the former ones are characterized by grid-shaped road networks, the latter ones rarely posses such regular geometric structures. Since the road networks in rural areas are much sparser than those of cities, we refrained from considering these networks and focused on the more complex city road networks. We extracted the abstract road graphs from the data provided by OpenStreetMap[1]. We applied the spherical Mercator projection ESPG:3857, which is also known as *Web Mercator* and used

[1] openstreetmap.org.

Table 1. Statistics for Baltimore (BA), Berlin (BE), Boston (BO), Los Angeles (LA), London (LO), Montreal (MO), Paris (PA), Rome (RO), Seattle (SE), Vienna (VI) and Washington (WA) for zoom 16. *OSM*: number of input segments in thousands. *Segm.:* percentage of segments after Phase 1, Step 3 in relation to input segments. *Graph*: number of road sections after Phase 1 in thousands. *Time:* running time for Phase 1.

	European cities					North American cities					
	BE	LO	PA	RO	VI	BA	BO	LA	MO	SE	WA
OSM	225.0	563.4	292.5	117.0	119.9	332.1	225.0	327.0	161.4	433.1	103.9
Segments	55	73	62	62	54	40	50	67	72	59	37
Graph	37.9	105.4	49.9	15.4	18.9	33.8	27.8	80.6	40.2	77.1	11.4
Time (sec.)	21	65	32	12	11	28	21	44	21	42	9

Table 2. *Speedup*: ratio of running times of two algorithms. *Quality:* ratio of the number of labeled road sections computed by two algorithms.

	Ratio	European cities					North American Cities						Avg.
		BE	LO	PA	RO	VI	BA	BO	LA	MO	SE	WA	
Speedup	$\frac{\text{MILP}}{\text{D\&C+MILP}}$	3.44	3.07	2.51	1.71	3.12	1.44	2.33	1.3	1.79	3.1	1.32	**2.29**
	$\frac{\text{TREE}}{\text{D\&C+TREE}}$	1.77	1.8	1.73	1.62	1.71	1.57	1.71	1.37	1.75	1.68	1.35	**1.64**
	$\frac{\text{D\&C+MILP}}{\text{D\&C+TREE}}$	2.82	2.32	3.33	2.54	2.74	6.84	3.06	21.59	6.36	5.32	10.59	**6.14**
Quality	$\frac{\text{D\&C+TREE}}{\text{TREE}}$	1.01	1.0	1.0	1.0	1.01	1.01	1.0	1.01	1.02	1.01	1.02	**1.01**
	$\frac{\text{D\&C+TREE}}{\text{MILP}}$	1.0	1.0	0.99	0.99	0.99	0.96	0.99	0.96	0.97	0.97	0.91	**0.97**
	$\frac{\text{Mapnik}}{\text{MILP}}$	0.74	0.85	0.83	0.91	0.76	0.71	0.8	0.62	0.61	0.8	0.68	**0.75**
	$\frac{\text{BASELINE}}{\text{MILP}}$	0.58	0.49	0.4	0.38	0.48	0.39	0.42	0.39	0.46	0.37	0.24	**0.42**
	$\frac{\text{D\&C+TREE}}{\text{Mapnik}}$	1.36	1.19	1.2	1.09	1.29	1.37	1.25	1.55	1.58	1.21	1.33	**1.31**

by several popular map-services. We considered the three scale factors 4.773, 2.387 and 1.193, which approximately correspond to the map scales 1:16000, 1:8000, 1:4000[2]. Further, they correspond to the *zoom levels* 15, 16 and 17, respectively, which are widely used by map services as OpenStreetMap. Those zoom levels show road networks in a size that already allows labeling single road sections, while the map is not yet so large that it becomes trivial to label the roads. We applied the standard drawing style for OpenStreetMap, which in particular includes the stroke width and color of roads as well as the font size of the labels. Further, this specifies for each zoom level the considered road categories; the higher the zoom level the more categories are taken into account.

Our implementation is written in C++ and compiled with GCC 4.8.4 using optimization level `-O3`. MILPs were solved by Gurobi[3] 6.0. The experiments were performed on a 4-core Intel Core i7-2600K CPU clocked at 3.4 GHz, with 32 GiB RAM. The D&C-approach labels single components in parallel. For computing the Delaunay triangulation we used the library Fade2d[4].

[2] wiki.openstreetmap.org/wiki/Zoom_levels.
[3] www.gurobi.com.
[4] www.geom.at.

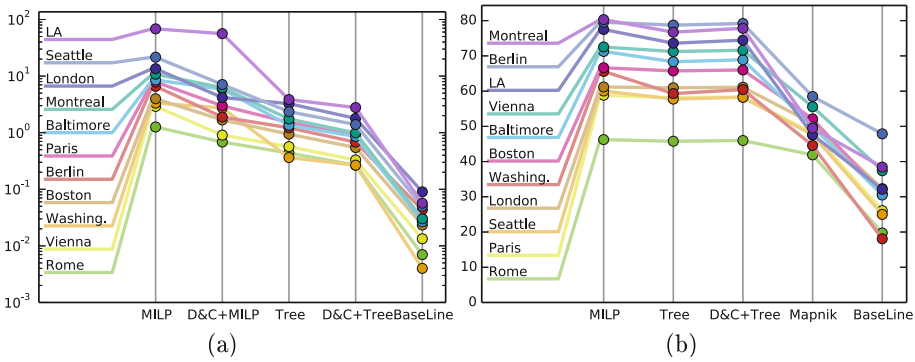

Fig. 5. (a) Running times in seconds of the algorithms (logarithmic scale). (b) Percentage of labeled road sections over all zoom levels broken into the different algorithms.

For each city and each zoom level we applied the algorithms BaseLine, Tree, D&C+Tree, Milp and D&C+Milp. We adapted the algorithm such that short road sections (shorter than the width of the letter W) are not counted, because they are rarely visible. Further, we let Mapnik (Version 3.0.9) render the same input. For each label we identified for each of its letters the closest road section r with the same name and counted it as labeled. Since Mapnik does not optimize the labeling by the same criteria as we do, we compensate this by also counting neighboring road sections as labeled if the junction in between them is not incident to any other road section. This accounts for those long road sections that we split artificially to resolve Issue 3.

The raw data of our experiments is made available on i11www.iti.kit.edu/roadlabeling. On this page we also provide interactive maps of the cities Berlin, London, Los Angeles and Washington, which present the computed labelings.

Phase 1. With a maximum of 67 s (London, zoom 17) and 27 s averaged over all instances, Phase 1 can be applied on large instances in reasonable time. During Phase 1 the number of segments is reduced to between 40 % and 83 % of the original instance (measured after Step 3, before creating junction edges); see also [7]. This clearly indicates that the procedure aggregates many lanes, since by design the approach does not change the overall geometry, but the simplification maintains the shape of the original network. This is also confirmed by the labelings; see Fig. 1(b)–(c) and interactive maps.

Phase 2, Running Time. We first consider the average running times over all zoom levels; see Fig. 5(a). We did not measure the running times of Mapnik, because its labeling procedure is strongly interwoven with the remaining rendering procedure, which prevents a fair comparison. As to be expected Milp is the slowest method (max. 126 s, Los Angeles, ZL 15), while BaseLine is the fastest procedure (max. 0.17 s). Combining Milp with D&C yields an average speedup of 2.29 over all instances and a max. speedup of 3.44; see Table 2.

The algorithm Tree needs less than 4.7 s and its median is about 1.3 s. Hence, despite its worst-case cubic asymptotic running time, it is fast in practice.

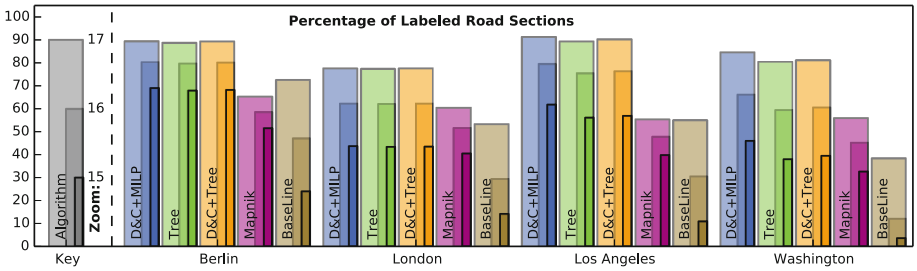

Fig. 6. Percentage of labeled road sections broken down in zoom levels and algorithms. The width of the bars (thin, medium, wide) corresponds to the zoom level (15, 16, 17).

Similar to MILP, it is further enhanced by combining it with D&C for a speedup of 1.64 with respect to TREE, and an average speedup of 6.14 with respect to D&C+MILP; see Table 2. In the latter case it even has a maximum speedup of about 21.6. Since decomposing and composing the labelings is done sequentially, the theoretically possible speed up using D&C is not achieved.

If we break down the running times into single zoom levels, we observe similar results; see also [7]. Since with increasing zoom level the instance size grows, for most of the algorithms also the running time increases. Only for North American cities and MILP do we observe that the running time for instances of smaller zoom levels are higher than for larger zoom levels.

Phase 2, Quality. First we analyze the average percentage of labeled road sections over the three zoom levels; see Fig. 5(b). As an upper bound, MILP, which provably solves MAXLABELEDROADS optimally, yields results from 46.2 % (Rome) to 80.3 % (Montreal). Considering zoom levels independently, we obtain a minimum of 27.5 % (Rome, ZL 15) and a maximum of 91.7 % (Montreal, ZL 17). We think that the wide span is attributed to the different structures of road networks and road names, e.g., Rome has a lot of short alleys and long road names. Hence, many road components are too short or convoluted to contain a single label. Abbreviating road names could help to overcome this problem.

The algorithm D&C+TREE yields marginally better results than TREE, but only 1 % on average, see Table 2. Comparing D&C+TREE with MILP we observe that D&C+TREE yields near-optimal results with respect to our road-section based model. On average it reaches 97 % of the optimal solution; see Table 2. While the quality ratio is only 91 % for Washington, more than half of the instances are labeled with a quality ratio of $\geq$99 %. For European cities the percentage of road sections that belong to components that are optimally solved by TREE (long edges, paths, and trees) is notably higher (88.3 %) than for North American cities (60.5 %). Nonetheless, we obtain similar percentages of labeled road sections for North American Cities. Hence, the heuristic computing a spanning tree of non-tree components is both fast and yields near-optimal results. The additional implementation effort of TREE is further justified by the observation that the naive way to place labels only on single road sections lags far behind;

only 42 % on average, 58 % as maximum and 24 % as minimum compared to the optimal solution. Mapnik achieves on average 75 % of the optimal solution and a maximum of 91 %. For more than the half of the instances Mapnik achieves at most 76 % of the optimal solution. So in direct comparison, D&C+Tree consistently outperforms Mapnik. Moreover, D&C+Tree has a better utilization of labels and achieves an average ratio of 1.61 labeled road sections per label, compared to Mapnik with a ratio of 1.37; see also [7].

With increasing zoom level the number of labeled road sections is increased, which is to be expected, since more road sections become long-edges; see Fig. 6 for four cities (similar results apply for the others). For each zoom level, we observe similar results as described before: Tree and D&C+Tree achieve near-optimal solutions and Mapnik labels considerably fewer road sections. However, for smaller zoom levels the gap between Milp and Mapnik shrinks.

From a visual perspective, labels lie on the skeleton of the road network, which is achieved by design; see Fig. 1(c) and the interactive maps. Instead of unnecessary repetition of labels, labels are only placed if they actually convey additional information. In particular, visual components are labeled, but not single lanes that are indistinguishable due to the zoom level.

6 Conclusion

We introduced a framework for labeling road maps based on an abstract road graph model that is combinatorial rather than geometric. We showed in our experimental evaluation that our proposed heuristic for decomposing the road graph into tree-shaped subgraphs and labeling those trees provably optimally is efficient and effective. It has running times in the range of seconds to one minute even for large road networks such as London with more than 100,000 road sections and achieves near-optimal quality ratios (on average 97 %) compared to upper bounds computed by the exact method Milp. Our algorithm clearly outperforms the labeling algorithm of the standard OSM renderer Mapnik, with an average improvement in the number of labeled road sections of 31 %. Interestingly, Milp is able to compute mathematically optimal solutions within a few minutes for all our test instances, even though it is slower by a factor of about 6 compared to the tree-based algorithm. So for practical purposes there is a trade-off between a final, but rather small improvement in quality at the cost of a significant and by the very nature of Milp unpredictable increase in running time. We only implemented essential cartographic criteria to evaluate the algorithmic core of our framework; further criteria (e.g., abbreviated names) and alternative definitions of road sections can be easily incorporated. The framework can be pipelined with labeling algorithms for other map features, e.g., after placing labels for point features, one may block all parts of the road network covered by a point label and label the remaining road network such that no labels overlap. While this allows the labeling of different types of features sequentially, constructing a labeling of all features in a single step remains an open problem.

Acknowledgment. We thank Andreas Gemsa for many interesting and inspiring discussions, and his help on the implementation.

References

1. Bader, M., Weibel, R.: Detecting and resolving size and proximity conflicts in the generalization of polygonal maps. In: International Cartographic Conference (ICC 1997), pp. 1525–1532 (1997)
2. Chirié, F.: Automated name placement with high cartographic quality: city street maps. Cartogr. Geogr. Inf. Sci. **27**(2), 101–110 (2000)
3. Gemsa, A., Niedermann, B., Nöllenburg, M.: Label placement in road maps. In: Paschos, V.T., Widmayer, P. (eds.) CIAC 2015. LNCS, vol. 9079, pp. 221–234. Springer, Heidelberg (2015)
4. Imhof, E.: Positioning names on maps. Am. Cartogr. **2**(2), 128–144 (1975)
5. Maass, S., Döllner, J.: Embedded labels for line features in interactive 3D virtual environments. In: Computer Graphics, Virtual Reality, Visualisation and Interaction (AFRIGRAPH 2003), pp. 53–59. ACM (2007)
6. Neyer, G., Wagner, F.: Labeling downtown. In: Bongiovanni, G., Petreschi, R., Gambosi, G. (eds.) CIAC 2000. LNCS, vol. 1767, pp. 113–124. Springer, Heidelberg (2000)
7. Niedermann, B., Nöllenburg, M.: An algorithmic framework for labeling road maps. CoRR. arXiv:1605.04265 (2016)
8. Reimer, A., Rylov, M.: Point-feature lettering of high cartographic quality: a multicriteria model with practical implementation. In: EuroCG 2014 (2014)
9. Schwartges, N., Wolff, A., Haunert, J.-H.: Labeling streets in interactive maps using embedded labels. In: Advances in Geographic Information Systems (ACM-GIS 2014), pp. 517–520. ACM (2014)
10. Seibert, S., Unger, W.: The hardness of placing street names in a Manhattan type map. Theor. Compt. Sci. **285**, 89–99 (2002)
11. Strijk, T.: Geometric algorithms for cartographic label placement. Dissertation, Utrecht University (2001)
12. Vaaraniemi, M., Treib, M., Westermann, R.: Temporally coherent real-time labeling of dynamic scenes. In: Computing for Geospatial Research Applications (COM.Geo 2012), pp. 17:1–17:10. ACM (2012)
13. van Kreveld, M.: Geographic information systems. In: Handbook of Discrete and Computational Geometry, 2nd edn., Chap. 58, pp. 1293–1314. CRC Press (2010)
14. Wolff, A., Strijk, T.: The map labeling bibliography (2009). http://liinwww.ira.uka.de/bibliography/Theory/map.labeling.html