# Practical Topologically Safe Rounding of Geographic Networks

Thomas C. van Dijk
Lehrstuhl für Informatik I
Würzburg University
thomas.van.dijk@uni-wuerzburg.de

Andre Löffler
Lehrstuhl für Informatik I
Würzburg University
andre.loeffler@uni-wuerzburg.de

## ABSTRACT

We consider the problem of accurately representing geographic networks at reduced coordinate precision. We require that vertices are placed on a grid and the network topology is retained, that is, we are not allowed to introduce intersections or collapse faces. Minimizing the "rounding error" in this setting is known to be $\mathcal{NP}$-hard and no practical methods, even heuristic, are known. We demonstrate a two-stage simulated annealing algorithm that focuses on finding a feasible solution first, then switches to optimizing the rounding error; a straightforward annealing approach without stage one has difficulty finding any feasible solution at all. We discuss various feasibility procedures and evaluate their applicability on geographic networks. Datasets and an implementation in C++ are available at: https://github.com/tcvdijk/armstrong.

## CCS CONCEPTS

• **Theory of computation → Simulated annealing**; *Data compression*; *Computational geometry*; • **Information systems** → *Geographic information systems.*

## KEYWORDS

graph algorithms, optimisation, simulated annealing

**ACM Reference Format:**
Thomas C. van Dijk and Andre Löffler. 2019. Practical Topologically Safe Rounding of Geographic Networks. In *27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '19), November 5–8, 2019, Chicago, IL, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3347146.3359347

## 1 INTRODUCTION

In this paper we consider the problem of representing the vertices of a geographic network at (integer) grid positions. There are several advantages to such representations, as opposed to the common practice of using floating-point numbers for coordinates. Firstly, this makes explicit what the actual precision of the representation is, in a data type without mathematical surprises [7]. Also, original coordinate precision has huge impact on the precision required to safely perform geometric operations, such as intersecting or calculating overlap [22]. Additionally, finding representations on particularly small grids is a natural form of data compression, since it reduces the range of the coordinate values. (It is also of mathematical interest to consider the smallest grid on which the network can be represented under certain quality constraints.) Furthermore, grid drawings also provide a form of schematization, by enforcing a minimum length on edges and introducing a rigid structure in dense areas. This also serves a perceptual purpose: there are no arbitrarily small features to be drawn. Such features would be hard to read and, ultimately, any visual reproduction of the network is likely to be discrete at some level anyway.

We are of course interested in *good* representations of the input network – for some measure of quality – not just an arbitrary representation. Many "rounding" and "snapping" procedures from the literature give a bound on the geometric difference between the input and output networks; usually, vertices are allowed to move only within one grid cell. They achieve this by accepting possible changes to the topology of the network, such as allowing vertices to coincide, new intersection to occur, or faces to collapse. We approach the problem from the other direction, by demanding topological equivalence between the input and output drawings and optimizing the quality of the result. This perspective is motivated by geographic networks, where connectivity, embedding (road networks), and the faces (territorial outline maps) are crucial. Our main measure of quality for the rounded output is as follows: the sum over the vertices, of the Euclidean distance between their input position and output position.

This topologically-safe "rounding" problem is nontrivial, especially if the network has areas where the density of the points is high relative to the size of the grid. In fact, several variants are known to be $\mathcal{NP}$-hard [20, 23] and no practical method for obtaining high-quality results is known. In this paper we present a practical method based on simulated annealing.

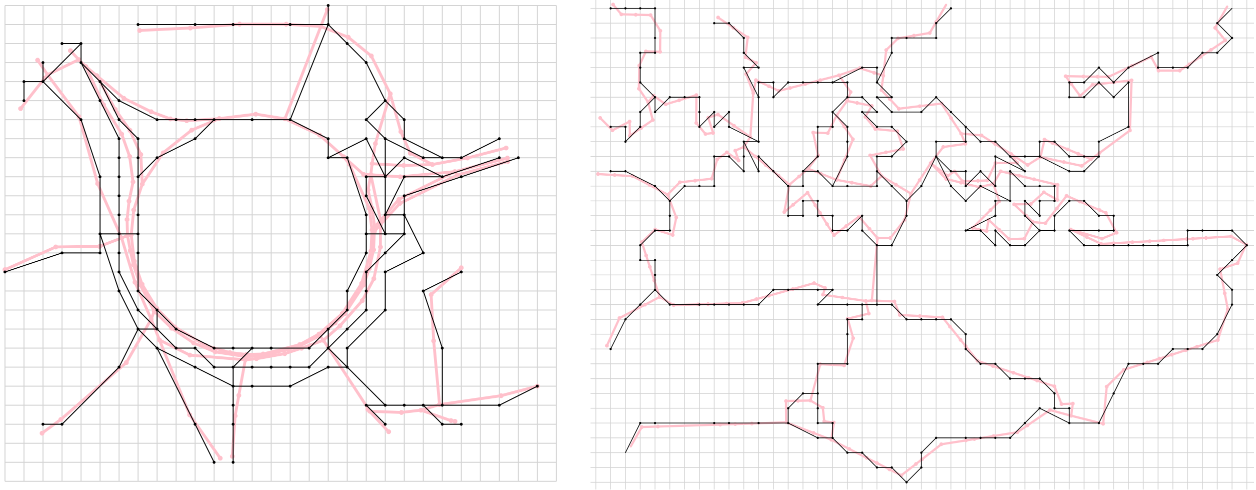
### 1.1 Related Work

Motivated by the limited resolution of raster-based computer graphics, Greene and Yao [9] introduced a framework for rounding line segments (endpoints and intersections) to integer coordinates. Their algorithm represents the vertices of a geometric graph on a grid, explicitly changing the network topology in order to achieve an error bound. This also addresses possible problems with the applicability of *real RAM* [27] algorithms, which are commonly used in computational geometry. Among many variants [2, 8, 13, 15, 16, 24], Guibas and Marimont coined *snap rounding* [12] for such algorithms. An implementation is available in CGAL [25].

From a computational geometry perspective it is possible to consider rounding specific classes of graphs, such as Voronoi diagrams [3]. In higher dimensions, mainly with polyhedra in mind, Goodrich et al. [8] propose a scheme for segments in 3D, and

*SIGSPATIAL '19, November 5–8, 2019, Chicago, IL, USA*

ACM ISBN 978-1-4503-6909-1/19/11…$15.00
https://doi.org/10.1145/3347146.3359347


(a)

(b)

**Figure 1: Two instances with real-valued input (light drawing) and corresponding grid representation (black drawing) computed using our algorithm. (a) Roundabout in downtown Würzburg (**$138$ **vertices,** $155$ **edges), grid size** $28 \times 28$**, average vertex movement** $0.600$**, computed in** $15\,s$**; (b) borders in Britain (**$3110$ **vertices,** $3207$ **edges), grid size** $240 \times 240$**, average vertex movement** $0.435$**, computed in** $70\,s$**, cropped to show only the south-east.**

Milenkovic [21] consider face lattices. Snap rounding of arrangements in 3D has recently been studied by Devillers et al. [4].

The above approaches bound the Hausdorff distance between input and output, but allow features to collapse. As argued before, these techniques may not be appropriate for geographic applications. Unfortunately, minimizing distortion in topologically-safe grid representations is $\mathcal{NP}$-hard in many settings [20]. Löffler et al. show that the following problem is $\mathcal{NP}$-hard: given a bounding box $\mathcal{B}$ and an embedded graph $G$ with real vertex positions, find a topologically equivalent drawing of $G$ with vertices at integer grid positions within $\mathcal{B}$, minimizing the sum of the Euclidean distances between each vertex's input and output positions.

Phrased as a problem of data compression, it is hard to find a minimum representation of arrangements of polygons [23]. Shekhar et al. [28] give a clustering-based approach to compress vector (road) maps; see Khot et al. [17] for a survey on road network compression techniques and challenges.

## 1.2 Our Contribution

In this paper, we consider the "topologically-safe snapping" problem [20] without bounding box. The hardness proofs from that paper trivially extend to our variant. Given a graph $G$ with vertex positions $p \colon V \longrightarrow \mathbb{R}^2$, we denote by $\Gamma(G, p)$ the subdivision of the plane induced a straight-line drawing of the graph. We call this a *drawing* of $G$. Let $\|\cdot\|$ be the Euclidean distance.

**Topologically-Safe Grid Representation**

*Instance:* Planar embedded graph $G = (V, E)$ with real vertex positions $r \colon V \longrightarrow \mathbb{R}^2$

*Output:* Vertex positions $p \colon V \longrightarrow \mathbb{N}^2$, such that $\Gamma(G, p)$ is topologically equivalent to $\Gamma(G, r)$, and $\sum_{v \in V} \|r(v) - p(v)\|$ is minimized.

Considering the hardness of this problem, we propose a heuristic approach – in particular, a two-staged algorithm based on simulated annealing. Stage one focuses on finding a feasible solution, since finding *any* topologically equivalent grid representation that does not, in the worst case, massively distort the input is nontrivial: the goal of this stage is to find a somewhat reasonable solution that we can improve in stage two. This second stage uses simulated annealing to improve the quality of the drawing. See Figure 1 for two example solutions.

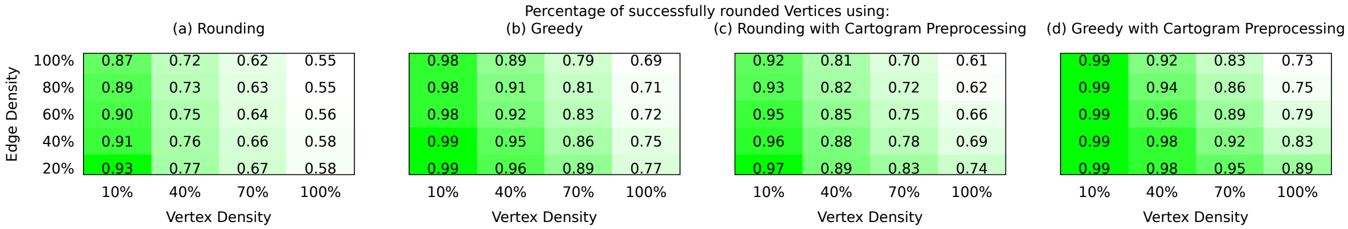
We have implemented this approach and evaluate it on real world road networks as well as on artificially generated networks. The latter allow us to statistically justify each major component of our algorithm. An implementation in C++ is available as open software.

## 2 TERMINOLOGY AND BASIC HEURISTICS

Call a vertex *nongrid* if its coordinates are not integer. The *cost* of a vertex $v$ in a drawing of $G$ is defined as the Euclidean distance between its original (real) position in the input and its position in the drawing; the cost of a drawing is the sum over the costs of its vertices. A drawing is called *feasible* if it is topologically equivalent to the input and all vertices are positioned on grid points. Note that the original drawing always has cost zero, but – except in trivial cases – is infeasible.

Since we will describe various iterative procedures, it will be useful to talk about *moving* a vertex: by that we mean changing its position while keeping the position of all other vertices unchanged. A move is called *valid* if it does not change the topology of the drawing, and the cost of the move is the difference in cost between the new and the old drawing.

As a baseline, we first describe several *partial* heuristics, in the sense that they are not guaranteed to find a feasible solution. By *rounding* a vertex, we mean moving it to the nearest grid point.



(a) Rounding

| Edge Density | 10% | 40% | 70% | 100% |
|---|---|---|---|---|
| 100% | 0.87 | 0.72 | 0.62 | 0.55 |
| 80% | 0.89 | 0.73 | 0.63 | 0.55 |
| 60% | 0.90 | 0.75 | 0.64 | 0.56 |
| 40% | 0.91 | 0.76 | 0.66 | 0.58 |
| 20% | 0.93 | 0.77 | 0.67 | 0.58 |

(b) Greedy

| Edge Density | 10% | 40% | 70% | 100% |
|---|---|---|---|---|
| 100% | 0.98 | 0.89 | 0.79 | 0.69 |
| 80% | 0.98 | 0.91 | 0.81 | 0.71 |
| 60% | 0.98 | 0.92 | 0.83 | 0.72 |
| 40% | 0.99 | 0.95 | 0.86 | 0.75 |
| 20% | 0.99 | 0.96 | 0.89 | 0.77 |

(c) Rounding with Cartogram Preprocessing

| Edge Density | 10% | 40% | 70% | 100% |
|---|---|---|---|---|
| 100% | 0.92 | 0.81 | 0.70 | 0.61 |
| 80% | 0.93 | 0.82 | 0.72 | 0.62 |
| 60% | 0.95 | 0.85 | 0.75 | 0.66 |
| 40% | 0.96 | 0.88 | 0.78 | 0.69 |
| 20% | 0.97 | 0.89 | 0.83 | 0.74 |

(d) Greedy with Cartogram Preprocessing

| Edge Density | 10% | 40% | 70% | 100% |
|---|---|---|---|---|
| 100% | 0.99 | 0.92 | 0.83 | 0.73 |
| 80% | 0.99 | 0.94 | 0.86 | 0.75 |
| 60% | 0.99 | 0.96 | 0.89 | 0.79 |
| 40% | 0.99 | 0.98 | 0.92 | 0.83 |
| 20% | 0.99 | 0.98 | 0.95 | 0.89 |

**Figure 2: Success rates of (a) incremental rounding, and (b) incremental greedy as described in Section 2. Figures (c) and (d) include the cartogram preprocessing from Section 3.3. Each entry is for 100 random instances with area $[0, 19] \times [0, 19]$ and varying vertex and edge density (see Section 4.1 for a description of the random instances).**

**Instant Rounding.** Independently set the position of each vertex to the nearest grid point. If the resulting drawing is topologically equivalent to the input, it is an optimal solution since every vertex individually moves the minimum amount possible. Otherwise the heuristic failed and does nothing.

**Incremental Rounding.** Round the vertices one by one, in arbitrary order, checking the validity of each move. Undo any invalid moves, leaving such vertices in their original position. In contrast to the instant rounding heuristic, this procedure may succeed in rounding a subset of the vertices even if it fails to solve the entire instance; this can be useful for quickly solving "easy" parts of the graph. The cost of this drawing is still a lower bound on the optimum, since any vertex that moves does so by the minimum amount.

**Incremental Greedy.** Do the following for all vertices, one by one, in arbitrary order: consider moving it to any of the four corners of the grid cell it is in.[1] Check these candidate moves for validity in nondecreasing order of cost (distance to $r(v)$); accept the first valid move and continue with the next vertex. Vertices with no valid candidates remain unrounded.

We also present a heuristic that always finds a feasible solution. However, it can give very bad solutions.

**Scale & Greedy.** Repeat the following until a valid drawing is found. Uniformly scale the input network by a factor $f$ (initially $f = 1$) and apply the Incremental Greedy algorithm. If all vertices are rounded we are done; else increase $f$ by one and retry. This process terminates, since Incremental Greedy will succeed for large enough $f$. However, scaling the entire network is likely to result in high cost.

Depending on the properties of the instances under consideration, these heuristics may work quite well or experience significant difficulties: see Figure 2 (a) and (b). Here we consider random planar graphs with various vertex densities and edge counts on a fixed area. These instances are described in more detail in Section 4.1; here we include vertex densities from 10% up to 100% (meaning one vertex per grid cell on average), and approximately matching-sized edge sets (20%) up to a full Delaunay triangulation (100%).

The matrices show the success rate of the Incremental Rounding and Incremental Greedy algorithms on 100 graphs for each setting: the fraction of *vertices* in the output drawings that are at grid position. We see that greedy clearly performs better than rounding, but even in the sparsest graphs, it rounds only 99% of the vertices on average. The gradient of the values indicates that vertex density seems to be more challenging for both approaches than the number of edges: although the edges also constrain the feasibility of drawings, it is particularly the (local) density of vertices that forms a problem for these heuristics. Note, for example, that a grid cell containing more than four input vertices cannot be completely rounded by these heuristics; this, among other difficult situations, is more likely to occur in denser instances. Scale & Greedy overcomes this problem by exploding the size of the network, effectively increasing the number of available grid points at the expense of a large (rounding) cost.[2] This is efficient in terms of runtime, but often requires large scale factors on real data and therefore returns drawings that are completely unacceptable as a heuristic and impractical for Stage Two. See also the experimental results of Section 4.2.1.

## 3 ALGORITHM

The difficulty of Topologically-Safe Grid Representation lies not just in finding a solution with low cost: it is nontrivial to find any decent feasible solution at all. Given these difficulties (and the intractability of finding optimal solutions [20]), we now consider a local search approach. Since we do not want to require starting with a feasible solution, this local search will have to handle infeasible states, that is, needs to consider drawings in which not all vertices lie on a grid point (yet). However, it never considers drawings that are not topologically equivalent to the input: that is an invariant we maintain at all times, rather than a property we are searching for. All our search algorithms use the following neighborhood.

**Local search neighborhood.** Take any vertex and perform any valid move among the following. If the vertex is already on the grid, move it to one of the eight grid points surrounding it. If the vertex is not on the grid, move it to a corner of the grid cell it is currently in.

If we use hill climbing (always greedily picking the neighbor with lowest cost) in this neighborhood definition, nothing happens: the input drawing already has cost zero and any other drawing is more expensive. Since we generally foresee further complications due

[1] This is ill defined for vertices on grid points and grid lines; in fact we take the floor or ceiling on each axis, giving *at most* 4 candidate positions for the vertex. In particular, this means that vertices that already lie on a grid point are not moved.

[2] From a data compression perspective, scaling by a factor two corresponds to using an additional bit on each axis of each vertex. With enough bits, the drawing can be represented without significant rounding.

to local optima, we pick the well-known metaheuristic *simulated annealing* [18]. For a description of simulated annealing, see for example Van Laarhoven and Aarts [32]; below we briefly sketch the general approach.

Simulated annealing is an iterative local search procedure. Consider all topologically valid drawings of the input network as possible *system states*. We define the *energy* (or: *cost*) of a state to be the rounding cost as defined in the problem statement. To transition between states, we pick a random (valid) move from the *neighborhood* described above. If the new state has lower cost than the current one, we *accept* it as our new current state. If the new state has higher cost – that is, it is worse – we can still accept it, and do so with probability $\exp(-\delta/T)$, where $\delta$ is the difference in cost and $T$ is a variable called the *temperature*. This allows simulated annealing to escape local optima. As *cooling schedule*, we employ the standard exponential schedule $T_n = c \cdot T_{n-1}$ for some constant $c$; in this way, it becomes less likely to accept worse solutions as the search progresses. We discuss the value of $c$ in Section 4.3.1.

We observe experimentally that a straightforward annealing approach using cost as the objective function does not perform well at all (see for example Figure 3): the focus on cost prevents it from finding a feasible drawing. It would be possible to design an objective function that rewards feasibility. However, this presents several difficulties. Firstly, in order for the search procedure to actually *find* the feasible drawings, this added term must be "smooth" enough to provide attraction, but it is not clear how to do this. Furthermore, depending on the details of this added term, it would have to be tweaked to not interfere with the cost optimization too much. We are eager to avoid such extra tuning parameters, which would come on top of the parameter tuning required for the simulated annealing itself. Therefore, we split the algorithm into two stages: one focused on quickly finding a reasonable feasible drawing, and a second straightforward annealing phase to minimize the cost.

## 3.1 Stage One – Feasibility

We describe several feasibility procedures. We first sketch an approach that is guaranteed to find a feasible drawing quickly. Unfortunately, these drawings are bad. Then we describe a local search procedure that works well in practice.

*3.1.1 Graph drawing.* Take the input as an abstract (embedded) graph and draw it anew, for example with the algorithm of Harel et al. [14] which – efficiently and deterministically – computes a compact grid drawing with a given embedding. Besides several technical challenges, such as requiring the graph to be biconnected, our experiments indicate that these Harel et al. drawings do not provide a good starting point for our Stage Two: they are too dissimilar to the input. See Section 4.2.2 for a qualitative evaluation of this approach.

*3.1.2 Vertex-density annealing.* This procedure uses the local-search neighborhood introduced before, but with a different objective function. Since locally dense regions are hard to successfully round, we want the search algorithm to make space. We therefore minimize the following objective function: the sum of inverse squared distances for all pairs of vertices.

$$f_{\text{density}}(p) = \sum_{v,\, w \in V,\, v \neq w} \frac{1}{\|p(v) - p(w)\|^2} \tag{1}$$

This is reminiscent of the repulsive forces in force directed graph drawing [6]. Note that we do not *actually* want to minimize this objective function (which is easy: scale the graph arbitrarily large). Rather, we run the search at a constant temperature of $T = 1$ and terminate the search as soon as the drawing is feasible. (This makes "annealing" a bit of a misnomer, but for uniformity of presentation, and since we do have Kirkpatrick-style move acceptance, we call it annealing.) We unconditionally accept any move that moves a nongrid vertex onto a grid point, regardless of its effect on the cost, since that is the real goal of this stage. The constant high temperature ensures enough freedom of movement: the goal of this stage is to escape locally difficult situations.

In order to find a feasible drawing sooner, we investigate two improvements to the neighbor selection. First, at each step, we perform the incremental greedy algorithm, immediately moving any nongrid vertices to the best available grid point on its cell. This shortcuts having to wait for the random vertex selection to pick such vertices eventually. Secondly, rather than selecting a vertex uniformly at random, we can select the vertex distributed according to its local density, that is, $\sum_{w \in V,\, w \neq v} 1/\|p(v) - p(w)\|^2$. This encourages the search algorithm in areas that are too dense, which hopefully gives progress toward feasibility. See Section 4.2.5 for a statistical evaluation validating that these are improvements.

*3.1.3 Grid-density annealing.* This functions identically to vertex-density annealing, except it interprets the density more locally based on the grid, using the following procedure. Every nongrid vertex adds 1/4 "density" to its four surrounding grid points and every other vertex adds 1/9 to its eight surrounding grid points and the one it sits on. Then we say the cost of a vertex is the squared density of the grid point it is on – or, for a nongrid vertex, the squared density of the nearest grid point. The annealing objective value is the sum of these vertex costs. As with Vertex-density annealing, this encourages vertices to move out of the way of other vertices, and particularly provide space for nongrid vertices (since they contribute more to the density). However, it is not clear if its more local nature is good or bad. This is evaluated in Section 4.2.3.

## 3.2 Stage Two – Reducing Cost

Once a feasible drawing has been found, we switch to straightforward simulated annealing with our real objective function: minimizing rounding cost. Since this is a real annealing approach where we want to avoid local optima, we pick the typical exponential cooling schedule. See Section 4.3 for details of the parameter selection.

## 3.3 Preprocessing using Linear Cartograms

Since dense areas of the input drawing are problematic, we propose the following preprocessing – before Stage One – based on efficient linear cartograms [30, 31]: we ask for any edges shorter than length $\sqrt{2}$ (the diagonal of a grid cell) to be elongated and for vertices that are too close together to be moved apart. We will see in Section 4.2.4 that this is quite effective in practice.

This is implemented using the following linear least-squares adjustment formulation, computing new positions $p$ given the original positions $r$. Note that this is an overconstrained set of equalities; least squares adjustment will find an optimal compromise. (See for example Kraus [19] for a general introduction.)

Firstly, in the interest of the cost of the drawing, vertices should ideally stay where they are. For every $v \in V$, we therefore have the following constraint on the $x$ and on the $y$ axis.

$$p(v) = r(v) \tag{2}$$

The relative position of vertices that are connected by an edge should also remain the same – except if that edge is very short: then we want to introduce some distance between these vertices. For every $(u, v) \in E$ and on both axes, we have either

$$p(v) = p(u) + r(v) - r(u) \quad \text{if } \|r(u) - r(v)\| > \sqrt{2}, \text{ or} \tag{3}$$

$$p(v) = p(u) + \frac{\sqrt{2} \cdot (r(v) - r(u))}{\|r(v) - r(u)\|} \tag{4}$$

otherwise, effectively requesting the edge to have length $\sqrt{2}$.

We add such constraints, with different weights, not only for edges of the input graph, but also for any pair of vertices that are within distance $\sqrt{2}$ ("too near") and for the edges of a constrained Delaunay triangulation (see [29] for more on constrained Delaunay triangulations when transforming geographic networks). We have experimentally observed the following weights to perform well, and use them throughout the paper: position 0.2, edges 4.0, too-near 2.0, Delaunay 1.0. (Experiments are omitted due to page limit.)

It is possible that the resulting drawing $\Gamma(G, p)$ is not topologically equivalent to the input. In that case, it is possible to use the event constraints of Haunert et al. [30]; instead, we use only their back-off procedure: consider linearly interpolating from the input drawing to the cartogram drawing, find the latest time in this interpolation that yields a valid drawing, and output that.

### 3.4 Postprocessing

Depending on the iteration count and the cooling schedule, Stage Two may produce drawings that are not locally optimal. We could anneal for longer – as the annealing temperature goes to zero, the search reduces to hill climbing – but a straightforward hill climbing implementation is much more efficient: rather than sampling random vertices and attempting moves, it iteratively applies improving, valid moves to all vertices until a local optimum is reached. This suggests the possibility of annealing at a higher temperature than one normally would, and relying on the final hill climb to clean up the solution; see Section 4.

### 3.5 Implementation Considerations

Our algorithm often needs to test whether a particular move is valid. The expensive part is checking for possibly intersecting edges. Rather than the well-known (and worst-case more efficient) Bentley-Ottmann sweepline algorithm [1] for line segment intersections, we implement the following grid-based approach. First, we overlay a $W \times W$ regular grid of rectangular bins over the full extent of the current drawing, then loop over all edges and put them in all bins that they intersect, and finally check all pairs of edges in each bin by brute force. If $W$ is picked large enough, at most a small number of edges will be in any particular bin. This is efficient on realistic data for a wide range of values $W$. Our code uses $W = 512$ as a somewhat arbitrary trade-off between memory, number of bins, and the population of bins. See also Peng and Wolff [26].

In fact, this grid-based approach – at least for our application and on our data – outperforms the CGAL implementation [35] of the Bentley-Ottmann algorithm by more than two orders of magnitude, even though CGAL generally has high-quality and high-performance implementations. Its problem seems to be the numerical instability of the sweepline algorithm, which requires CGAL to use high-precision arithmetic on our real-world data: something our relatively crude algorithm does not require. We additionally point out that our current implementation is single threaded, but in principle can be easily parallelized.

We use CGAL for basic geometric computations and for constrained Delaunay triangulations [34]. We use Eigen [10] for highly-efficient sparse matrix maths in the cartogram code.

## 4 EXPERIMENTAL RESULTS

In this section, we statistically evaluate various properties of our algorithm and the effectiveness of various design decisions. Whenever we directly compare two options, we provide a (two-sided) Wilcoxon paired signed-rank test [33] and report the $z$-score, demonstrating statistically significant improvements; recall that a $z$-score of 1.96 or higher satisfies a 95% confidence level.

### 4.1 Test Instances

We provide experiments on both real world networks and artificial instances. For the real networks, we have used GeoFabrik's OpenStreetMap shapefiles[3] and the City of Chicago Open Data Portal.[4] Since these road networks are not always planar, we have introduced vertices at any intersections in order to get plane graphs.

We also consider random instances in order to perform systematic and statistically significant experiments. These are based on binomial point processes, placing $v$ vertices uniformly at random in a square area of $X$ by $X$ units. To generate instances with many edges, we take the Delaunay triangulation [11] of this point set; for instances with fewer edges, we take a subset of those edges as described below. We therefore have three parameters: the side length $X$ of the area in which the points are sampled, the number of vertices $v$, and the number of edges as a fraction $\varepsilon$ of Delaunay triangulation. As an alternative to specifying the number of vertices $v$, we can also consider the ratio $\gamma$ between the number of vertices and the number of grid points of the area (up to rounding $v = \gamma \cdot (X + 1)^2$). In the rest of the section, we describe random instances by the triple of these parameters $(X, \gamma, \varepsilon)$.

Note that by calculating a Delaunay triangulation of the point set, we can also determine a value of $\gamma$ and $\varepsilon$ for real instances. The road networks we consider have $\varepsilon \in [35\%, 45\%]$ – see Table 1. In order to get these reduced edge densities from our Delaunay instances without introducing isolated vertices (which our algorithm does not consider), we proceed as follows. We sample the point set and calculate a Delaunay triangulation. Then we take a perfect matching in an arbitrary DFS tree of this triangulation (which exists [5]) marking those edges; we delete a uniformly random set of the other

[3] https://www.geofabrik.de/data/download.html

[4] https://data.cityofchicago.org/

edges to achieve the desired number of edges. Since the marked edges are a perfect matching, this leaves no isolated vertices.

## 4.2 Evaluating Stage One

As alluded to before, the basic rounding heuristics will not consistently find feasible drawings and those provided by the graph drawing algorithm are too distorted for the annealing process to find a good solution in reasonable time. In this section, we evaluate the performance of the procedures from Section 3.1 on artificial instances. As a baseline, we first look at the most basic form, that is, without greedy steps during the annealing and sampling the vertices uniformly. Tthe experiments in this section are run on 1000 random instances of (19, 40%, 100%), that is, 400 grid points in the sampled area and therefore 160 vertices and about 450 edges.

To decide between the various procedures, we first rule out Scale & Greedy and Redrawing; then we compare Vertex-density annealing to Grid-density annealing, with and without cartogram preprocessing.

*4.2.1 Scale & Greedy is bad.* The instances in this experiment admit reasonable solutions with a cost of about 500 (as we will see). Scale & Greedy required an average scale factor of 3.86 (between 2 and 23), resulting in an average cost of 3329 and no solution with cost below 1047. This holds true also for real-world instances, producing a feasible drawing of the roundabout in Würzburg from Fig. 1 (a) requires a scaling factor of 4. This disqualifies Scale & Greedy as a practical Stage One procedure.

*4.2.2 Redrawing is worse.* We created grid drawings of all 1000 instances using the algorithm of Harel et al. [14]. The average cost of these drawings is an enormous 26256 (or average cost per vertex of 164). This is completely impractical: consider for example that any move in Stage Two can improve the cost by at most $\sqrt{2}$.

*4.2.3 Vertex density vs. Grid density.* We have proposed two different "density" objective functions and claimed that immediately optimizing cost has difficulty finding feasible solutions. We now experimentally evaluate this.

We ran the three options on each graph for 20000 iterations (after finding a feasible solution, the remaining iterations were spent optimizing cost). See Figure 3 for the behavior on a typical instance: the purple line shows cost, the turquoise line shows the number of vertices that have taken a grid positions. Cost annealing did not find a feasible solution for any of the instances. A general trend over the 1000 instances is that Vertex-density annealing generally requires fewer steps to find a feasible solution, whereas Grid-density annealing generally finds a (first) feasible solution with lower cost. Indeed, a Wilcoxon test shows that this is significant: Vertex-density annealing significantly finishes sooner ($z \approx 26$, "winning" 698 of the 1000 instances), but with higher cost ($z \approx 2.466$). As a qualitative indication, when it finished first, the Grid-density solution was 10% cheaper on average (363.794 compared to 406.019). On the instances where Vertex-density was faster, the average cost of the solutions found was higher: 490.993. This suggests that Grid-density is quicker at finding relatively easy solutions, but struggles on more difficult instances.

*4.2.4 Cartogram preprocessing.* When applying the cartogram preprocessing from Section 3.3, the claims from the previous section become even more significant: Vertex-density annealing now finishes first with $z \approx 38$, winning on 777 instances. The preprocessing also reduces the number of iterations used by either type of annealing: it makes Vertex-density faster with $z \approx 29$ and Grid-density with $z \approx 18$. In terms of the cost of the feasible solution found, the improvement for Vertex-density annealing falls short of significance ($z \approx 1.847$). For Grid-density annealing, preprocessing did result in better solutions ($z \approx 6.199$).

This demonstrates that preprocessing instances using linear edge cartograms is highly advisable in practice; it is fast – in our implementation, it takes the time of about 50 annealing iterations on typical instances – and improves the process of finding a feasible initial solution in almost every aspect.

*4.2.5 Incremental greedy and Nonuniform sampling.* Finally we add the two additional features from Section 3.1.2: nonuniform sampling of the vertex to move, and greedy steps. Still on the same instances, we observe the following. Vertex-density is still faster than Grid-density (winning 665 instances); the average cost for Grid-density solutions is still lower, on both winning and losing instances. Most notably, however, in winning instances of both variants, the cost of the solutions found are distinctly lower: 239 on average for Grid-density and 287 for Vertex-density annealing. Areas of high density in the input drawing require the vertices to be moved further – and thus more often – than in sparser regions. Sampling by density makes complicated vertices get picked more often, resolving conflicts faster and thereby inducing less distortion on other parts of the network. The incremental greedy steps are also quite cheap and can potentially save many iterations of annealing (which is also good for the cost). Consider also that moving a single vertex can allow multiple other vertices to be moved to grid points greedily.

## 4.3 Evaluating Stage Two

Now that we demonstrated how to find a reasonable initial solution, we consider Stage Two. After Stage One tried to get away from the input drawing to find a feasible drawing, Stage Two now anneals back towards the initial vertex positions. Annealing theory [18] suggests that a system provided with the right cooling schedule and enough time will eventually end up in a low cost state. It is unclear how to determine that this state is reached, and a schedule that is guaranteed to achieve it with high probability is impractically slow; in practice, this requires experimental tuning.

We note that our moves from drawing to drawing are rather small, in terms of objective value: a single vertex is moved at most one grid cell, resulting is a maximum change in cost of $\sqrt{2}$. With the typical starting at temperature of $T_0 = 1.0$, this means we initially accept at least 24.3% of all cost-increasing moves. We will evaluate various cooling schedules, step counts on large instances, and discuss when to switch from annealing to the hill climbing postprocessing as described in Section 3.4.

Finally, we note that unfortunately we cannot compare to optimal solutions for any interesting instances: known exact methods are wildly infeasible and only able to handle networks so small that the comparison is not very interesting.

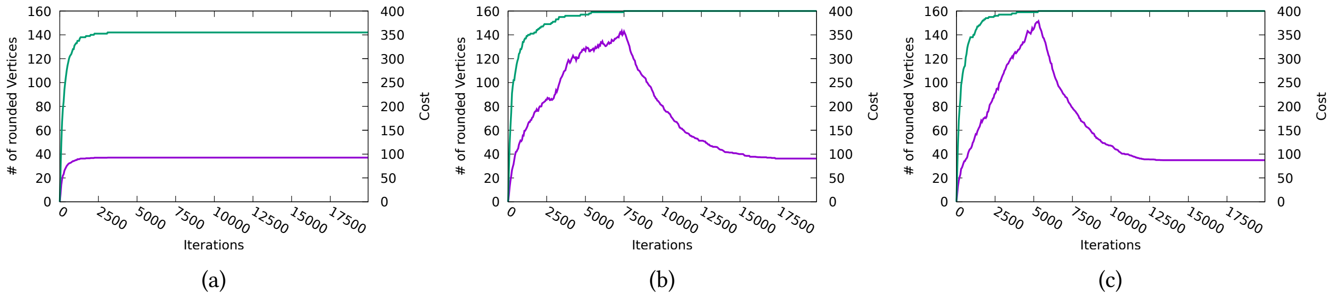


**Figure 3: Evaluating Stage One performance on one random instance; annealing Stage One for: (a) cost, (b) grid density, and (c) vertex density. Purple line shows total rounding cost, turquoise line number of rounded vertices.**

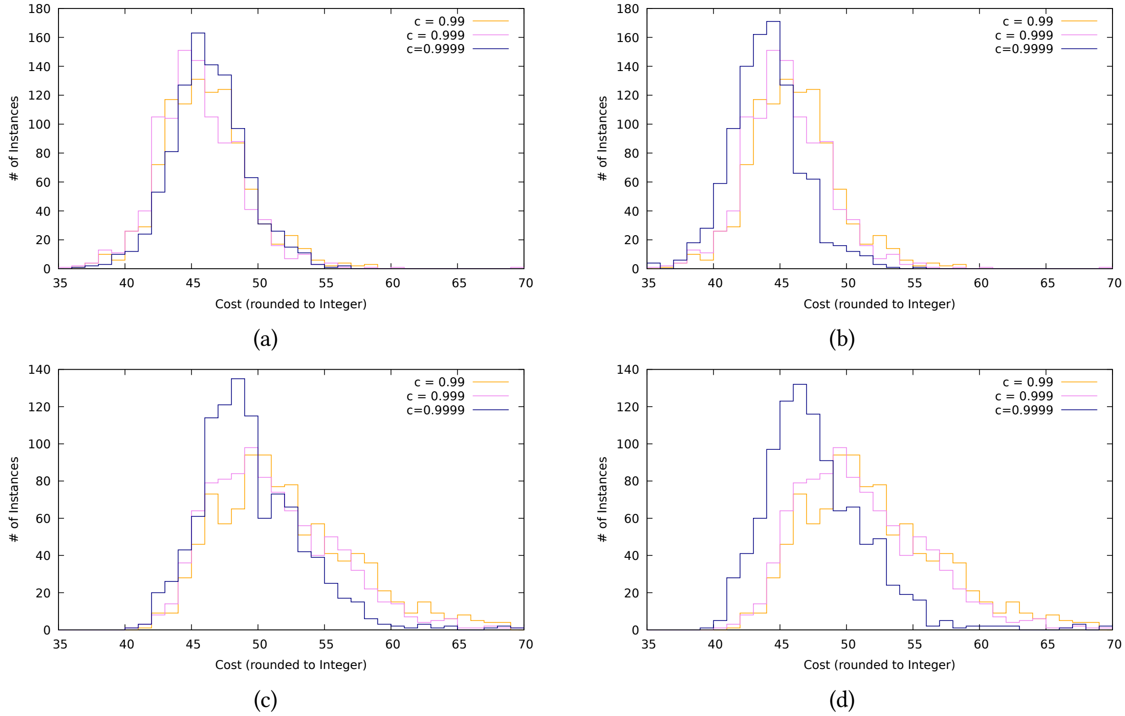


**Figure 4: Effect of cooling factor on cost: cost of $(14, 40\%, 40\%)$ instances (a) before / (b) after hill climbing; cost of $(14, 40\%, 100\%)$ instances (c) before / (d) after hill climbing. (Single outliers beyond cost of 70 ignored.)**

*4.3.1 Cooling Schedules.* We use the typical exponential cooling schedule $T_{i+1} = c \cdot T_i$, requiring a choice for initial temperature $T_0$ and cooling factor $c$. We evaluate this on 200 instances with parameters $(14, 40\%, 40\%)$ and 200 with $(14, 40\%, 100\%)$. In order to eliminate any randomness from re-running Stage One, we use precomputed feasible solutions using Vertex-density annealing. We ran 20000 steps in Stage Two (followed by hill climbing postprocessing) with $c_1 = 0.99$, $c_2 = 0.999$, and $c_3 = 0.9999$, doing five runs of each setting on each instance. Figure 4 shows a histogram of the resulting cost (rounded to the nearest integer). First consider subfigures (a) and (b), the rather sparse instances: before hill climbing (a), the choice of cooling factor does not seem to have huge impact on the quality; afterward (b), the advantage of the slower cooling provided by $c = 0.9999$ becomes visible. The reason is as follows: the schedule for $c_1$ reaches a temperature of effectively 0 after only 1500 iterations, compared to 12000 for $c_2$. This means that for both parameter values, a significant amount of time was spent rejecting any move that does not strictly improve the cost: they were basically hill climbing random vertices one step at a time. This is also visible in the plots: there is hardly any difference between the orange and red lines in (a) and (b). This is not true for the slowest schedule ($c_3$), as even after 20000 iterations the temperature was still about 0.135, leaving room for improvements to be made by hill climbing. The same phenomenon can be observed looking at the second set of instances (Figure 4 (c) and (d)). Again, hill climbing basically did not find any improvements for $c_1$ or $c_2$, whereas $c_3$ managed to avoid running into local optima long enough for Hill Climbing to make a difference. The cost improvement of hill climbing on both test sets are highly significant ($z > 50$ each).

With these observations in mind, we recommend always using the hill climbing as postprocessing; it is fast and guaranteed to not worsen the final solution – performing a set of moves that would have been accepted even at temperature 0.

*4.3.2 Step Count & Hill Climbing.* We generate feasible solutions for all of the (19, 40%, 100%) instances from Section 4.2 using Vertex-density annealing with cartogram preprocessing. On these rather large and dense instances, we run Stage Two annealing for $m$ steps ($m \in \{0, 2500, 5000, 10000, \ldots, 40000\}$), followed by hill climbing, reporting score with and without postprocessing. To demonstrate the impact of step count, we set the cooling schedule to $c = 0.9999$. Indicative results are shown in Figure 5.

As we are dealing with dense instances, resulting in rather expensive Stage One solutions (see Section 4.2.3), leaving space for a lot of improvements (we can expect "good" final solutions to have cost below 100). As noted before, annealing with temperature 0 yields very similar results to hill climbing. After 5000 steps, the system is still at temperature 0.607, whereas after 40000 steps, it reaches a final temperature of 0.0183.

A downside of combining a slow cooling schedule with greedy local optimization is visible in Figure 5 (a); we spent 2500 iterations randomly moving away from the first feasible solution, to which hill climbing sometimes could not repair.

Figure 5 (b) compares $m = 15000$ steps with hill climbing to $m = 25000$ steps without: doing more steps in Stage Two leads to lower costs in the final drawing. However, picking the point at which to switch from annealing to hill climbing carefully can lead to similar results in shorter time.

Figure 5 (c) shows the best results in terms of cost. Even after 40000 iterations of annealing, hill climbing managed to make some progress on 874 of the instances. There is also still a noticeable difference to lower step counts.

Figure 5 (d) illustrates impact hill climbing can have on solution costs. The improvement made by hill climbing is obvious and easy to explain – Stage Two simply was not done yet. Nevertheless, the final result is comparable to those of higher iteration counts.

## 4.4 Real-World Data

In this section we compare real-world instances to artificial networks and evaluate our approach further. These experiments here were performed on a 2.6 GHz processor with sufficient RAM. See Table 1 for experimental results; more data on more instances is available at github.com/tcvdijk/armstrong.

The instances were processed with Vertex-density annealing and all options enabled. Examine the instances and computed grid representations found in Figure 6 – (a) Würzburg–Train Station and (b) Chicago–Cloud Gate. The solutions of Würzburg–Train Station we computed had an average cost of 110.9 (0.860 on average per vertex, standard deviation of 8.66); notice that for these runs, we picked the vertex-grid point ratio $\gamma = 16.45\%$. The average cost over 100 runs on a random instance resembling this real network (by picking $\varepsilon$ accoringly, but with $\gamma = 40\%$) is 77.2 (0.483 on average per vertex, standard deviation of 1.92). While Würzburg–Train Station is less dense and thus could be expected to be easier, neither cost nor runtime reflect this; Train Station takes 63% longer on average (43.1 $s$ vs. 26.4 $s$) and is 43% more expensive. To investigate this, consider that the left part of Fig 6 (a) (marked by the blue arrow) is significantly more dense than the rest; moving the tilted bus parking lanes to the grid results in parts of them being pushed outwards. The vertex indicated by the blue arrow (together with the middle of its incident edges) force significant distortion in this drawing. While there seem to be quite some empty grid points nearby, topology prohibits any local improvement for the vertex marked by the orange arrow and on closer reflection, this seems to be a decent drawing of this instance on a critically small grid. To support this observation, we extracted the dense part: Würzburg–Bus Lanes alone at $\gamma = 40\%$ is comparable to artificial instances of about twice its size on runtime and rounding cost.

While the roads around Chicago–Cloud Gate in Fig. 6 (b) are generally represented quite well (and would have tolerated an even coarser grid), the curvature of the footpath on the bottom right is represented with way too many vertices to reasonably fit on the provided grid. The average vertex cost after Stage One (3.62 per vertex) indicates how long it took to find any feasible drawing. This suggests that future work could attempt to integrate topologically-safe simplification into the grid representation workflow.

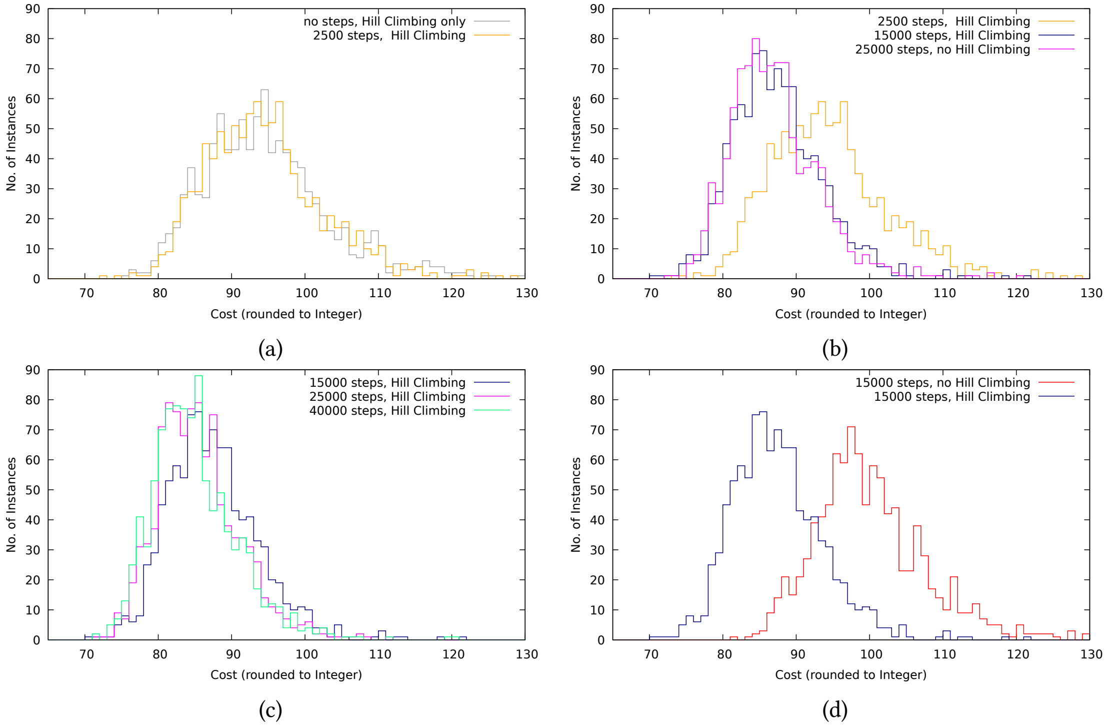
Again in Table 1, consider the artifical instances 0.4_0.4_42 and 0.4_0.4_84: different networks with the same parameters. The former is immediately solved by greedy steps, so the result of stage one is deterministic. (This is only the case when cartogram preprocessing is enabled, demonstrating its benefit.) Though the average runtime on the two instances is comparable, the latter's has higher variance. This suggests that a deterministic stage one may be preferable: the results obtained from stage two become are more stable.

All this indicates that the performance of our algorithm is sensitive to the structure of the network, and that the particulars of road network representations in OpenStreetMap and the City of Chicago Data Open Data Portal are challenging instances. Still, our method is able to find reasonable representations of realistic networks, which was not feasiable with previous methods.

# 5 CONCLUSION & FUTURE WORK

We have presented a practical heuristic for the Topologically-Safe Grid Representation problem, which is $\mathcal{NP}$-hard. The various features of the algorithm have been statistically validated as improving the runtime or the solution quality, and an open source implementation is available at https://github.com/tcvdijk/armstrong.

We have considered the grid representation problem in isolation, as a somewhat abstract problem. Future work can consider the place of grid representations in a larger context. For example, we noted it may be useful to integrate polyline simplification for geographic data. It would also be interesting to evaluate the influence that rounding to a grid representation has on subsequent steps in a pipeline, such as the length of shortest paths, the results of generalization, or, for example, map matching.



(a)

(b)

(c)

(d)

**Figure 5: Step count $m$ and the effect on cost of $(19, 40\%, 100\%)$ instances with and without hill climbing: (a) $m = 0$ vs. $m = 2500$, both hill climbing; (b) Hill climbing vs. more steps; (c) $m \in \{15000, 25000, 40000\}$, with hill climbing; (d) $m = 15000$, with and without hill climbing. Outliers have been cropped; for comparison, line colors carry over.**

| Name | | $v$ | $\varepsilon$ | grid size | S2 steps | runtime | $(\sigma_t)$ | S1 cost | $(\sigma_{S1})$ | S2 cost | $(\sigma_{S2})$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Würzburg | –Train Station | 129 | 40.3% | 28 × 28 | 20000 | 43.2 *s* | (14.9 *s*) | 486.2 | (119.9) | 110.4 | (7.20) |
| | –Bus Lanes | 89 | 41.7% | 15 × 15 | 20000 | 23.9 *s* | (4.73 *s*) | 242.0 | (59.7) | 72.6 | (6.23) |
| | –Ring | 138 | 38.8% | 20 × 20 | 20000 | 60.7 *s* | (9.54 *s*) | 392.5 | (67.2) | 115.4 | (4.94) |
| Chicago | –Downtown | 358 | 35.2% | 25 × 25 | 20000 | 83.3 *s* | (18.3 *s*) | 1462.7 | (106.8) | 390.4 | (16.0) |
| | –Cloud Gate | 578 | 35.1% | 240 × 240 | 35000 | 492 *s* | (119.4 *s*) | 2090.8 | (313.1) | 356.6 | (8.69) |
| United Kingdom | –States | 3110 | 34.8% | 250 × 250 | 50000 | 590 *s* | (282.3 *s*) | 5817.3 | (938.5) | 1344.2 | (11.1) |
| Artificial | –0.4_0.4_42 | 160 | 40% | 20 × 20 | 20000 | 26.4 *s* | (0.88 *s*) | 135.7 | (0) | 77.2 | (2.15) |
| | –0.4_0.4_84 | 160 | 40% | 20 × 20 | 20000 | 25.6 *s* | (1.50 *s*) | 205.9 | (50.4) | 77.9 | (3.18) |
| | –0.4_1.0_42 | 160 | 100% | 20 × 20 | 20000 | 63.7 *s* | (11.5 *s*) | 454.6 | (181.9) | 94.7 | (10.1) |

**Table 1: Selected test instances (real and artificial) including effective parameter values relating to the artificial instances. The instances were processed using cartogram preprocessing, Vertex Density Annealing for stage one, and hill climbing postprocessing. All numbers presented here are the average over 100 runs with the same parameters. "Runtime" is given in seconds, single-threaded; "S2 steps" is the number of iterations in stage two; "S1 cost" and "S2 cost" are total rounding cost after stages one and two respectively; $\sigma_t, \sigma_{S1}$ and $\sigma_{S2}$ are the corresponding standard deviations on the measurements.**

## REFERENCES

[1] Bentley and Ottmann. 1979. Algorithms for Reporting and Counting Geometric Intersections. *IEEE Trans. Comput.* C-28, 9 (sep 1979), 643–647. https://doi.org/10.1109/tc.1979.1675432

[2] Mark de Berg, Dan Halperin, and Mark Overmars. 2007. An intersection-sensitive algorithm for snap rounding. *Computational Geometry* 36, 3 (apr 2007), 159–165. https://doi.org/10.1016/j.comgeo.2006.03.002

[3] Olivier Devillers and Pierre-Marie Gandoin. 2002. Rounding Voronoi diagram. *Theoretical Computer Science* 283, 1 (jun 2002), 203–221. https://doi.org/10.1016/s0304-3975(01)00076-7

[4] Olivier Devillers, Sylvain Lazard, and William J. Lenhart. 2018. 3D Snap Rounding. (2018). https://doi.org/10.4230/lipics.socg.2018.30

[5] Michael B. Dillencourt. 1987. A non-Hamiltonian, nondegenerate Delaunay Triangulation. *Inform. Process. Lett.* 25, 3 (may 1987), 149–151. https://doi.org/10.1016/0020-0190(87)90124-4

[6] P. Eades. 1984. A heuristic for graph drawing. *Congressus Numerantium* 42 (1984), 149–160.

[7] David Goldberg. 1991. What every computer scientist should know about floating-point arithmetic. *Comput. Surveys* 23, 1 (mar 1991), 5–48. https://doi.org/10.1145/103162.103163

[8] Michael T. Goodrich, Leonidas J. Guibas, John Hershberger, and Paul J. Tanenbaum. 1997. Snap rounding line segments efficiently in two and three dimensions.

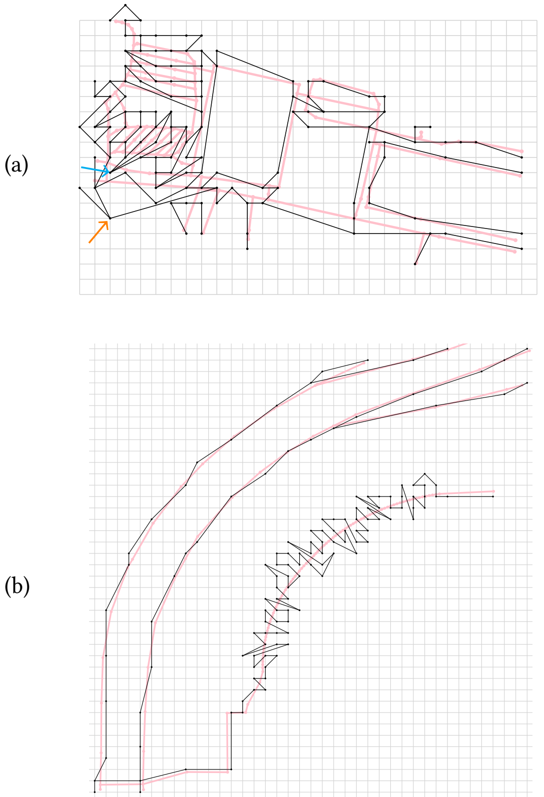
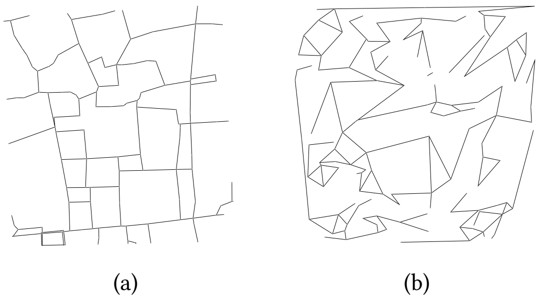


**Figure 6: Grid representations of real-world instances with problematic features. (a) Würzburg–Train Station on $28 \times 28$ in 20000 steps. Note the vertex indicated by the orange arrow: it looks highly suboptimal. However, consider its outgoing edge to the right. Also note the vertex indicated by the blue arrow, and how few alternatives it has; (b) a crop of Chicago–Cloud Gate on 240×240 in 40000 steps. Note that the rightmost path has way too many vertices compared to the grid size.**



(a) (b)

**Figure 7: Visual comparison of real-world and artificial networks: (a) Würzburg downtown with 134 vertices and $\varepsilon = 40.5\%$; (b) an artificial instance with the same parameters.**

In *Proceedings of the 13th Annual Symposium on Computational Geometry*. ACM, 284–293. https://doi.org/10.1145/262839.262985

[9] Daniel H. Greene and F. Frances Yao. 1986. Finite-resolution computational geometry. In *27th Annual Symposium on Foundations of Computer Science*. IEEE, 143–152. https://doi.org/10.1109/sfcs.1986.19

[10] Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org. (2010).

[11] Leonidas J. Guibas, Donald E. Knuth, and Micha Sharir. 1992. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica* 7, 1-6 (jun 1992), 381–413. https://doi.org/10.1007/bf01758770

[12] Leonidas J. Guibas and David H. Marimont. 1998. Rounding Arrangements Dynamically. *International Journal of Computational Geometry & Applications* 8, 02 (1998), 157–178. https://doi.org/10.1142/s0218195998000096

[13] Dan Halperin and Eli Packer. 2002. Iterated snap rounding. *Computational Geometry* 23, 2 (2002), 209–225. https://doi.org/10.1016/s0925-7721(01)00064-5

[14] D. Harel and M. Sardas. 1998. An Algorithm for Straight-Line Drawing of Planar Graphs. *Algorithmica* 20, 2 (feb 1998), 119–135. https://doi.org/10.1007/pl00009189

[15] John Hershberger. 2013. Stable snap rounding. *Computational Geometry* 46, 4 (2013), 403–416. https://doi.org/10.1016/j.comgeo.2012.02.011

[16] John D. Hobby. 1999. Practical segment intersection with finite precision output. *Computational Geometry* 13, 4 (1999), 199–214. https://doi.org/10.1016/s0925-7721(99)00021-8

[17] Amruta Khot, Abdeltawab Hendawi, Anderson Nascimento, Raj Katti, Ankur Teredesai, and Mohamed Ali. 2014. Road network compression techniques in spatiotemporal embedded systems. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on GeoStreaming - IWGS '14*. ACM Press. https://doi.org/10.1145/2676552.2676645

[18] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* 220, 4598 (may 1983), 671–680. https://doi.org/10.1126/science.220.4598.671

[19] Karl Kraus. 2011. *Photogrammetry: geometry from images and laser scans*. Walter de Gruyter.

[20] Andre Löffler, Thomas van Dijk, and Alexander Wolff. 2016. Snapping Graph Drawings to the Grid Optimally. In *Proceedings of the 26th International Symposium on Graph Drawing*. Springer, 144–151. https://doi.org/10.1007/978-3-319-50106-2_12

[21] Victor J Milenkovic. 1990. Rounding face lattices in d dimensions. In *Proc. 2nd Canad. Conf. Comput. Geom.* Citeseer, 40–45.

[22] Victor J Milenkovic. 1995. Practical methods for set operations on polygons using exact arithmetic.. In *CCCG*. Citeseer, 55–60.

[23] Victor J Milenkovic and Lee R Nackman. 1990. Finding compact coordinate representations for polygons and polyhedra. *IBM Journal of Research and Development* 34, 5 (1990), 753–769.

[24] Eli Packer. 2006. Iterated snap rounding with bounded drift. In *Proceedings of the 22nd Annual Symposium on Computational Geometry*. ACM, 367–376. https://doi.org/10.1145/1137856.1137910

[25] Eli Packer. 2019. 2D Snap Rounding. In *CGAL User and Reference Manual* (4.14 ed.). CGAL Editorial Board. https://doc.cgal.org/4.14/Manual/packages.html#PkgSnapRounding2

[26] Dongliang Peng and Alexander Wolff. 2014. Watch Your Data Structures!. In *Proc. 22nd Annual Conference of the GIS Research UK*.

[27] Micheal Ian Shamos. 1978. *Computational Geometry*. Yale University.

[28] Shashi Shekhar, Yan Huang, Judy Djugash, and Changqing Zhou. 2002. Vector map compression. In *Proceedings of the tenth ACM International Symposium on Advances in Geographic Information Systems - GIS '02*. ACM Press. https://doi.org/10.1145/585147.585164

[29] Thomas van Dijk, Arthur van Goethem, Jan-Haunert Haunert, Wouter Meulemans, and Bettina Speckmann. 2013. Accentuating focus maps via partial schematization. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - SIGSPATIAL'13*. ACM Press. https://doi.org/10.1145/2525314.2525452

[30] Thomas C. van Dijk and Jan-Henrik Haunert. 2014. Interactive focus maps using least-squares optimization. *International Journal of Geographical Information Science* 28, 10 (mar 2014), 2052–2075. https://doi.org/10.1080/13658816.2014.887718

[31] Thomas C. van Dijk and Dieter Lutz. 2018. Realtime linear cartograms and metro maps. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - SIGSPATIAL '18*. ACM Press. https://doi.org/10.1145/3274895.3274959

[32] Peter J. M. van Laarhoven and Emile H. L. Aarts. 1987. *Simulated Annealing: Theory and Applications*. Springer Netherlands. https://doi.org/10.1007/978-94-015-7744-1

[33] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (dec 1945), 80. https://doi.org/10.2307/3001968

[34] Mariette Yvinec. 2019. 2D Triangulation. In *CGAL User and Reference Manual* (4.14 ed.). CGAL Editorial Board. https://doc.cgal.org/4.14/Manual/packages.html#PkgTriangulation2.

[35] Baruch Zukerman, Ron Wein, and Efi Fogel. 2019. 2D Intersection of Curves. In *CGAL User and Reference Manual* (4.14 ed.). CGAL Editorial Board. https://doc.cgal.org/4.14/Manual/packages.html#PkgSurfaceSweep2.