# On $d$-regular Schematization of Embedded Paths

Daniel Delling[a], Andreas Gemsa[b], Martin Nöllenburg[b], Thomas Pajor[b], Ignaz Rutter[b]

[a]*Microsoft Research Silicon Valley, Mountain View, CA, USA*
[b]*Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany*

## Abstract

Motivated by drawing route sketches, we consider the $d$-regular path schematization problem. For this problem we are given an embedded path $P$ (e.g., a route in a road network) and a positive integer $d$. The goal is to find a *$d$-schematized* embedding of $P$ in which the orthogonal order of all vertices in the input is preserved and in which every edge has a direction that is an integer multiple of $(90/d)^\circ$. We show that deciding whether a path can be $d$-schematized is NP-complete for any positive integer $d$.

Despite the NP-hardness of the problem we still want to be able to generate route sketches and thus need to solve the $d$-regular path schematization problem. We explore two different algorithmic approaches, both of which consider two additional quality constraints: We require that every edge is drawn with a user-specified minimum length and we want to maximize the number of edges that are drawn with their preferred direction. The first algorithmic approach restricts the input paths to be axis-monotone. We show that there exists a polynomial-time algorithm that solves the $d$-regular path schematization problem for this restricted class of embedded paths. We extend this approach by a heuristic such that it can handle arbitrary simple paths. However, for the second step we cannot guarantee that the orthogonal order of the input embedding is maintained. The second approach is a formulation of the $d$-regular path schematization problem as a mixed-integer linear program. Finally, we give an experimental evaluation which shows that both approaches generate reasonable route sketches for real-world data.

*Keywords:* Path Schematization, NP-hardness, Mixed Integer Linear Programming, Dynamic Programming, Graph Drawing
*2010 MSC:* 68U05, 68Q17, 68R10

Preliminary versions of the results presented in this paper have appeared at the 12th Scandinavian Symposium and Workshop on Algorithm Theory (SWAT'10) [1], and at the 37th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'11) [2].

The corresponding author of this submission is Andreas Gemsa (gemsa@kit.edu).

Part of this work was done within GRADR – EUROGIGA project no. 10-EuroGIGA-OP-003. A.G. and M.N. are financially supported by the *Concept for the Future* of KIT within the framework of the German Excellence Initiative.

*Email addresses:* `dadellin@microsoft.com` (Daniel Delling), `gemsa@kit.edu` (Andreas Gemsa), `noellenburg@kit.edu` (Martin Nöllenburg), `pajor@kit.edu` (Thomas Pajor), `rutter@kit.edu` (Ignaz Rutter)

## 1. Introduction

Angular or $\mathcal{C}$-oriented schematizations of graphs refer to a class of graph drawings, in which the admissible edge directions are limited to a given set $\mathcal{C}$ of (usually evenly spaced) directions. This includes the well-known class of orthogonal drawings and extends more generally to $k$-linear drawings, e.g., octilinear metro maps. Applications of schematic drawings can be found in various domains such as cartography, VLSI layout, and information visualization.

In many schematization scenarios the input is not just a graph but a graph with an initial drawing that has to be schematized according to the given set of directions. This is the case, e.g., in cartography, where the geographic positions of network vertices and edges are given [3], in sketch-based graph drawing, where a sketch of a drawing is given and the task is to improve or schematize that sketch [4], or in dynamic graph drawing, where each drawing in a sequence of drawings must be similar to its predecessor [5]. For such a redrawing task it is crucial that the mental map [6] of the user is preserved, i.e., the output drawing must be as similar as possible to the input. Misue et al. [6] suggested preserving the *orthogonal order* of the input drawing as a simple criterion for maintaining a set of basic spatial properties of the input, namely the relative above/below and left/right positions of all pairs of input vertices. The orthogonal order has been used successfully as a means for maintaining the mental map [7, 8, 9, 1].

The motivation behind the work presented here is the visualization of routes in road networks as sketches for driving directions. An important property of a route sketch is that it focuses on road changes and important landmarks rather than exact geography and distances. Typically the start and destination lie in populated areas that are locally reached via a sequence of relatively short road segments. On the other hand, the majority of the route typically consists of long highway segments with no or only few road changes. This property makes it difficult to display driving directions for the whole route in a single fixed-scale map since some areas require much larger scales than others in order to be readable. The strength of route sketches for this purpose is that they are not drawn to scale but rather use space proportionally to the route's complexity.

*Related Work.* Geometrically, we can consider a route to be an embedded path in the plane. The simplification of paths (or polylines) in cartography is well studied and the classic line simplification algorithm by Douglas and Peucker [10] is one of the most popular methods. Two more recent algorithms were proposed for $\mathcal{C}$-oriented line simplification [11, 12]. These line simplification algorithms, however, are not well suited for drawing route sketches since they keep the positions of the input points fixed or within small local regions around the input points and thus edge lengths are more or less fixed. On the other hand, Agrawala and Stolte [13] presented a system called *LineDrive* that uses heuristic methods based on simulated annealing to draw route sketches. A problem similar to drawing route sketches is generating destination maps, where given a destination (e.g. a restaurant) the aim is to produce a schematized map that helps anyone within a given region around the destination to reach it. Kopf et al. consider this problem [14] and give a a heuristic algorithm based on the continuation method. The methods of both of these approaches allow distortion of edge lengths and angles. However, they do not restrict the set of edge directions and do not give hard quality guarantees for the mental map such as the preservation of the orthogonal order.

A graph drawing problem that has similar constraints as drawing route sketches is the metro-map layout problem, in which an embedded graph is to be redrawn octilinearly. The problem is known to be NP-hard [15] but it can be solved successfully in practice by mixed-integer linear programming [16]. The existing methods, covered in a survey by Wolff [17], do aim to keep the mental map of the input, but no strict criterion like the orthogonal order is applied. Brandes and Pampel [9] studied the path schematization problem in the presence of orthogonal order constraints in order to preserve the mental map. They showed that deciding whether a rectilinear schematization exists that preserves the orthogonal order of the input path is NP-hard. They also showed that schematizing a path using arbitrarily oriented unit-length edges is NP-hard. Recently, the authors have published more simplified version of both proofs [18]. In [19] Speckmann and Verbeek consider the problem of finding a rectilinear, homotopic schematization of a collection of paths with a minimum number of path segments. The problem turns out to be NP-hard, but the authors give an approximation algorithm. Verbeek extends the approach in [20] to $\mathcal{C}$-oriented paths, where the path segments in the schematization can have only orientations that are contained in $\mathcal{C}$. However, besides the homotopy requirement, no special emphasis is placed on maintaining the user's mental map. Another type of $\mathcal{C}$-oriented schematization that has applications in generating schematized maps is studied by Buchin et al. [21]. There, the authors give an algorithm that finds a $\mathcal{C}$-oriented schematization of a polygon, where the polygon's edges must have directions that are contained in $\mathcal{C}$ and where the schematized polygon preserves the area of the input polygon.

*Contributions.* Motivated by drawing route sketches, we investigate the problem of generating $\mathcal{C}$-oriented orthogonal-order preserving drawings. We prove that deciding whether a $\mathcal{C}$-oriented orthogonal-order preserving drawing of an embedded input path exists is NP-complete, even if the path is simple. This is true for every *d-regular* set $\mathcal{C}$ of directions that have angles that are integer multiples of $90°/d$ for any integer $d$. This extends the result of Brandes and Pampel [9, 18] who showed NP-completeness for the case $d = 1$. However, their proof relies on the absence of diagonal edges and hence does not extend to larger values of $d$. We show the NP-completeness in the octilinear case $d = 2$ in Section 3 and, subsequently in Section 4, how this result extends to the general $d$-regular case for $d > 2$. In Section 5 we show that if we restrict the input to *axis-monotone* paths, i.e., $x$- or $y$-monotone paths, the $d$-regular path schematization problem can be solved in polynomial time. We present an efficient algorithm that finds, for a given embedded axis-monotone path, an embedding that uses only $d$-regular directions and maintains the orthogonal order of the input path. The algorithm maximizes the number of edges that are embedded with their preferred direction, i. e., the $d$-regular direction that is closest to the direction in the input embedding. We also heuristically extend this approach to non-monotone paths. In Section 6 we design a mixed integer-linear program (MIP) for solving the $d$-regular path schematization problem for arbitrary paths. Finally, in Section 7 we give an experimental evaluation of both the MIP approach and the heuristic simple path schematization algorithm. The source code of our proof-of-concept implementation in C++ is made available at `http://i11www.iti.kit.edu/_media/projects/routesketch.zip`. We summarize our results in Section 8.

## 2. Preliminaries

A *plane embedding* of a graph $G = (V, E)$ is a mapping that maps every vertex $v \in V$ to a distinct point $\pi(v) = (x_\pi(v), y_\pi(v))$ and every edge $e = uv \in E$ to the line segment $\pi(e) = \overline{\pi(u)\pi(v)}$ such that no two edges $e_1, e_2$ cross in $\pi$ except at common endpoints. For simplicity we also use the terms vertex and edge to refer to their images under an embedding, and, unless stated otherwise, we assume every embedding to be a plane embedding. We denote the length of an edge $e$ in $\pi$ as $|\pi(e)|$.

Let $\pi$ and $\rho$ be two embeddings of the same graph $G$. We say that $\rho$ preserves the *orthogonal order* [9] of $\pi$ if for any two vertices $u$ and $v \in V$ it holds that $x_\pi(u) \le x_\pi(v) \Rightarrow x_\rho(u) \le x_\rho(v)$ and $y_\pi(u) \le y_\pi(v) \Rightarrow y_\rho(u) \le y_\rho(v)$. In other words, the orthogonal order defines the relative above-below and left-right positions of any two vertices.

For an input embedding $\pi$ we measure the direction $\alpha_\pi(uv)$ of an edge $uv$ as the counterclockwise angle formed between the horizontal line through $\pi(u)$ and the line segment $\pi(uv)$. For a set of angles $\mathcal{C}$ we say that a drawing is *$\mathcal{C}$-oriented* if the direction of every edge $e \in E$ is contained in $\mathcal{C}$. A set of directions $\mathcal{C}$ is called *$d$-regular* for an integer $d$ if $\mathcal{C} = \mathcal{C}_d = \{(i \cdot 90°/d) \bmod 360 \mid i \in \mathbb{Z}\}$. Note that it appears that the orientation of an edge is for our problem irrelevant with the implication that the direction 45° and 225° are in principle identical. This is true for the NP-completeness proofs in Sections 3 and 4, but for our efficient algorithm for schematizing axis-monotone paths (see Section 5) this is different. Assuming the axis-monotone path is $x$-monotone, then the set $\mathcal{C}$ can contain arbitrary directions, as long as it also contains $0°, 90°, 270°$ as well as at least one diagonal direction strictly between 0° and 90° and at least one diagonal direction strictly between 270° and 360°.

Let $(G, \pi)$ denote a graph $G$ with a plane input embedding $\pi$. For brevity we may refer to the tuple $(G, \pi)$ as (embedded) graph. A *$d$-regular schematization* (or *$d$-schematization*) of $(G, \pi)$ is a plane embedding $\rho$ that is $\mathcal{C}_d$-oriented, that preserves the orthogonal order of $\pi$ and where no two vertices are embedded at the same coordinates. We also call $\rho$ *valid* if it is a $d$-schematization of $(G, \pi)$.

We denote by $\omega_\mathcal{C}(e)$ the *preferred direction* $\gamma \in \mathcal{C}$ of an edge $e$, i. e., the direction in $\mathcal{C}$ that is closest to $\alpha_\pi(e)$. For a $\mathcal{C}$-oriented embedding $\rho$ of $P$ and an edge $e \in P$ there is a *direction cost* $c_\rho(e)$ that captures by how much the direction $\alpha_\rho(e)$ deviates from $\omega_\mathcal{C}(e)$. More specifically, we say the direction cost of an edge

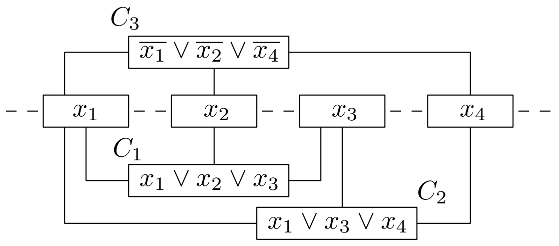


Figure 1: A Planar Monotone 3-Sat instance with four variables and three clauses.

is 1 if it is not embedded with its preferred direction and 0 otherwise. Note that there are edges which need to be embedded horizontally (vertically) to respect the orthogonal order of the input. The *schematization cost* $c(\rho)$ is then defined as $c(\rho) = \sum_{e \in P} c_\rho(e)$.

Finally, a *monotone path decomposition* for a given embedded path $(P, \pi)$ is a decomposition $\mathcal{P} = \{(P_1, \pi_1), \dots, (P_k, \pi_k)\}$ of $(P, \pi)$ into consecutive axis-monotone embedded subpaths $(P_i, \pi_i)$ for $1 \leq i \leq k$, where $\pi_i$ equals $\pi$ restricted to the subpath $P_i$.

## 3. NP-completeness of 2-regular Path Schematization

In this section we show that the problem of deciding whether there is a 2-schematization for a given embedded graph $(G, \pi)$ is NP-complete, even if $G$ is a simple path. In the latter case we denote the problem as the ($d$-regular) Path Schematization Problem (PSP). We fix $d = 2$. To prove that 2-regular PSP is NP-complete we first show hardness of the closely related 2-regular Union of Paths Schematization Problem (UPSP), where $(G, \pi)$ is a set $\mathcal{P}$ of $k$ disjoint embedded paths $\mathcal{P} = \{P_1, \dots, P_k\}$. We then extend this proof to a single path and conclude with a proof that the problem is in NP.

We show that 2-regular UPSP is NP-hard by a reduction from Planar Monotone 3-Sat, which is known to be NP-hard [22]. Planar Monotone 3-Sat is a special variant of Planar 3-Sat where each clause either contains exactly three positive literals or exactly three negative literals and additionally, the variable-clause graph admits a planar drawing such that all variables are on the $x$-axis, the positive clauses are embedded below the $x$-axis and the negative clauses above the $x$-axis. An example instance of Planar Monotone 3-Sat is depicted in Figure 1. In a second step, we show how to augment the set of paths $\mathcal{P}$ to form a single simple path $P$ that has the same properties as $\mathcal{P}$ and thus proves that PSP is NP-hard. This two-step approach is similar to the way Brandes and Pampel proved NP-hardness for the case $d = 1$ [9]. They also first proved NP-hardness for a set of paths and then extended the proof to show hardness for a union of paths. However, it seems highly unlikely that it is possible to adapt the gadgets in their proof (neither from the original proof [9] nor from the simplified version [18]) to our version of the problem. Their gadgets rely on being able to force with the embedding of a single edge another edge to be embedded either vertically or horizontally. This seems very difficult to achieve when allowing diagonal directions.

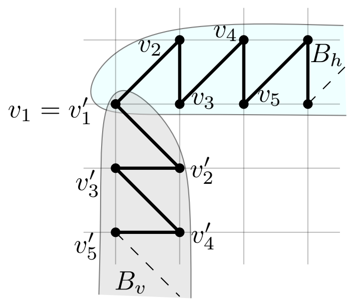


Figure 2: Border gadget $B$ that is rigid and rational. The horizontal component is denoted by $B_h$ and the vertical one by $B_v$.

In the following, we assume that $\varphi$ is a given Planar Monotone 3-Sat instance with variables $X = \{x_1, \dots, x_n\}$ and clauses $C = \{c_1, \dots, c_m\}$.

## *3.1. Hardness of the Union of Paths Schematization Problem*

We begin by introducing the different types of required gadgets and then show how to combine them in order to prove the hardness of UPSP.

*Border Gadget.* For all our gadgets we need to control the placement of vertices on discrete positions. Thus our first gadget is a path whose embedding is unique up to scaling and translation. This path will induce a grid on which we subsequently place the remaining gadgets.

Let $\Delta_x(u, v)$ denote the $x$-distance between the two vertices $u$ and $v$. Likewise, let $\Delta_y(u, v)$ be the $y$-distance between $u$ and $v$. We call a simple path $P$ with embedding $\pi$ *rigid* if a 2-schematization of $(P, \pi)$ is unique up to scaling and translations. Further, we call an embedding $\pi$ of $P$ *rational* if there exists a length $\ell > 0$ such that for any two vertices $u, v$ of $P$ it holds that $\Delta_x(u, v) = z_x \cdot \ell$ and $\Delta_y(u, v) = z_y \cdot \ell$ for some $z_x, z_y \in \mathbb{Z}$. Thus in a rational embedding of $P$ all vertices are embedded on a grid whose cells have side length $\ell$. For our border gadget we construct a simple path $B$ of appropriate length with embedding $\pi$ that is both rigid and rational. Hence, $\pi$ is essentially the unique 2-schematization of $(B, \pi)$, and after rescaling we can assume that all points of $B$ lie on an integer grid. The border gadget consists of a horizontal component $B_h$ and a vertical component $B_v$ which share a common starting vertex $v_1 = v'_1$; see Figure 2. The component $B_h$ alternates between a 45° edge to the upper right and a vertical edge downwards. The vertices are placed such that their $y$-coordinates alternate and hence all odd, and all even, vertices have the same $y$-coordinate, respectively. The vertical component consists of a copy of the horizontal component rotated by 90° in clockwise direction around $v_1$. To form $B$, we connect $B_v$ and $B_h$ by identifying their starting points $v_1$ and $v'_1$.

**Lemma 1.** *The border gadget $B$ with its given embedding $\pi$ is rigid and rational.*

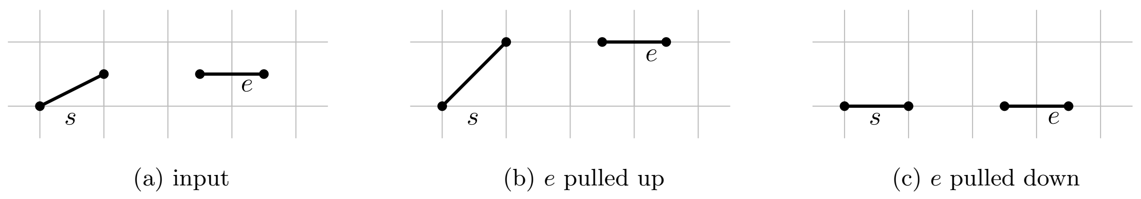


Figure 3: The switch $s$ is linked with edge $e$; (a) shows the input embedding, in (b) $e$ is pulled up and in (c) it is pulled down.

*Proof.* First note that in $B_h$ due to the orthogonal order all vertices $v_i$ with $i$ odd and all vertices $v_i$ with $i$ even have the same $y$-coordinates, respectively, in any valid schematization. Hence $\Delta_y(v_i, v_{i+1})$ is the same for all $i$. We show that $v_i$ and $v_{i+1}$ for $i$ odd, also have the same $x$-distance $\Delta_x(v_i, v_{i+1}) = \Delta_y(v_i, v_{i+1})$.

Since the edge $v_1v_2$ connects two vertices with different $y$-coordinates and since $v_2v_3$ must be embedded vertically, $v_1v_2$ must be embedded with an angle of 45°. And hence we have $\Delta_x(v_1, v_2) = \Delta_y(v_1, v_2)$. The same argument holds for all edges $v_iv_{i+1}$ with $i$ odd and hence $B_h$ is both rational and rigid.

Since the vertical border gadget $B_v$ is a copy of $B_h$ it is also rational and rigid. Moreover, we have that $\Delta_x(v'_1, v'_2) = \Delta_x(v_1, v_2)$, i.e., the distances between vertices of $B_v$ that lie on the same horizontal or vertical lines are the same as for $B_h$. Hence the border gadget $B$ is rigid and rational. □

In the following we define the length of a grid cell induced by $B$ as 1 so that we obtain an integer grid. We choose $B$ long enough to guarantee that any vertex of our subsequent gadgets lies vertically and horizontally between two pairs of vertices in $B$.

The grid in combination with the orthogonal order gives us the following properties:

1. If we place a vertex $v$ with two integer coordinates, i.e., on a grid point, its position in any valid embedding is unique;
2. if we place a vertex $v$ with one integer and one non-integer coordinate, i.e., on a grid edge, its position in any valid embedding is on that grid edge;
3. if we place a vertex $v$ with two non-integer coordinates, i.e., in the interior of a grid cell, then its position in any valid embedding is in that grid cell (including its boundary).

*Basic Building Blocks.* We will frequently make use of two basic building blocks that rely on the above grid properties.

The first one is a *switch*, i.e., an edge that has exactly two valid embeddings. Let $s = uv$ be an edge within a single grid cell where $u$ is placed on a grid point and $v$ on a non-incident grid edge. We call $u$ the *fixed* and $v$ the *free* vertex of $s$. Assume that $u$ is in the lower left corner and $v$ on the right edge of the grid cell. Then in any valid $d$-schematization $s$ has one of only two possible embeddings, either $s$ is embedded horizontally or diagonally. This property justifies the name switch; see Figure 3 for an example.

The second basic concept is *linking* of vertices. We can synchronize two vertices in different and even non-adjacent cells of the grid by assigning them the same $x$- or $y$-coordinate. We call two vertices $u$ and $v$ *linked*, if in $\pi$ either $x_\pi(u) = x_\pi(v)$ or $y_\pi(u) = y_p(v)$. Then the orthogonal order requires that $u$ and $v$ remain linked in any valid embedding. This concept allows us to transmit information on local embedding choices over distances. Two edges $e_i = uu'$ and $e_j = vv'$ are *linked* if there is a vertex of $e_i$ that is linked to a vertex of $e_j$. We use linking of edges in combination with switches. Namely, we link a switch $s$ via its free vertex with another edge $e$, as illustrated in Figure 3. Then the choice of the embedding of $s$ determines one of the two coordinates of the linked vertices in $e$. In the case depicted in Figure 3 the switch $s$ determines the $y$-coordinate of both vertices of $e$; we say that $s$ *pulls* $e$ up (Figure 3(b)) or down (Figure 3(c)). Such a switch is called a *vertical switch*. Analogously, edges can be pulled to the left and to the right by a *horizontal switch*.

In the following, if we say that an edge or vertex can only be embedded in a certain way, we always mean that the edge or vertex can only be embedded this way in any valid schematization. Now that we have the switches and the concept of linking available, we will combine them to high-level constructions that model variables and clauses of a Planar Monotone 3-Sat formula.

*Variable Gadgets.* The variable gadget for a variable $x$ is a simple structure consisting of a horizontal switch and a number of linked *connector vertices* on consecutive grid lines below the switch, one for each appearance of $x$ or $\neg x$ in a clause of $\varphi$. We denote the number of appearances as $t(x)$. All connector vertices share the same $x$-coordinate in $\pi$. Each one will be connected to a clause with a diagonal *connector edge*. A connector edge spans the same number of grid cells horizontally and vertically and hence can only be embedded at a direction of 135°. Since both vertices of a connector edge are placed on grid lines the connector edge can only have correct direction if the positions of both vertices within their respective grid cells is the same. The upper connector vertices of the variable gadget connect to the gadgets of the negative clauses and the lower connector vertices to the gadgets of the positive clauses. The variable gadget has two states, one in which the switch pulls all vertices to the left (defined as `true`), and one in which it pulls them to the right (defined as `false`). Figure 4 shows an example. A variable gadget $g_x$ for a variable $x$ takes up one grid cell in width and $t(x) + 1$ grid cells in height, one grid cell for each connector vertex and an additional grid cell for the horizontal switch.

*Clause Gadgets.* The general idea behind the clause gadget is to create a set of paths whose embedding is influenced by three connector edges. Each of these edges carries the truth value of the connected variable gadget. The gadget consists of three diagonal edges $e_2, e_3, e_4$, two vertical edges $e_1$ and $e_5$ and four switches $s_1, s_2, s_3, s_4$; see Figure 5. For positive clauses, we want that if all its connector edges are pulled to the right, i.e., all literals are `false`, there is no valid embedding of the clause gadget. This is achieved by placing the edges of the gadget in such a way that there are certain points, called *critical points*, where in a valid

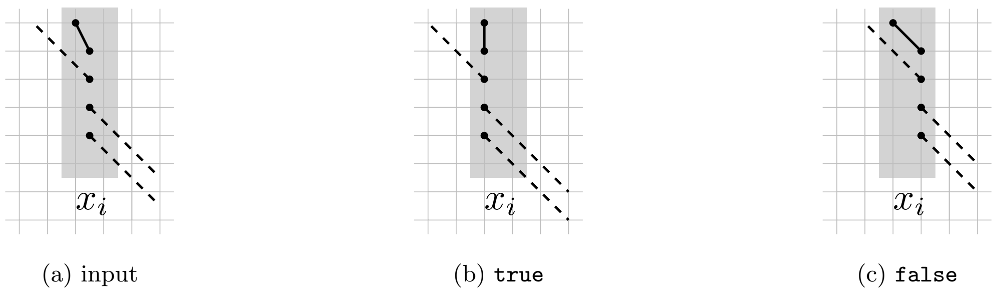


Figure 4: (a) The input embedding of a variable gadget with three connector vertices, (b) the state `true`, and (c) the state `false`.

embedding two different non-adjacent edges can possibly have a vertex. Further, we ensure with the help of switches that each of the three diagonal edges of the clause gadget has exactly one vertex embedded on a critical point in a valid embedding.

Key to the gadget is the *critical edge* $e_3$ that is embedded with one fixed vertex on a grid point and one loose vertex inside the grid cell $A$ to the top left of the fixed vertex. Edge $e_3$ is linked with the vertical switch $s_2$ and the horizontal switch $s_4$. These switches ensure that the loose vertex must be placed on a free corner of the grid cell $A$. The three free corners are all critical points. It is clear that there is a valid embedding of $e_3$ if and only if at least one of the three critical points is available.

We use the remaining edges of the gadget to block one of the critical points of $A$ for each literal that is `false`. The lower vertex of the middle connector edge is placed just to the left of the upper left corner of $A$ such that it occupies a critical point if it is pulled to the right. Both, the left and right connector edges have another linked vertical edge $e_1$ and $e_5$ appended. Now, if $e_1$ is pushed to the right by its connector edge, then the edge $e_2$ is pushed upward since $e_1$ and $e_2$ share a critical point. Due to switch $s_1$ edge $e_2$ blocks the lower left critical point of $A$ in that case. Similarly, edge $e_4$ blocks the upper right critical point of $A$ if the third literal is `false`.

The clause gadget for a negative clause works analogously such that a critical point is blocked for each connector edge that is pulled to the left instead of to the right. It corresponds basically to the positive clause gadget rotated by 180°.

**Lemma 2.** *A clause gadget has a valid embedding if and only if at least one of its literals is* `true`*.*

*Proof.* To see this we note that there are five critical points in the gadget and the connector edge of each literal that is `false` blocks one of the critical points. The three edges $e_2, e_3, e_4$ must also occupy one critical point each. So if all literals are `false`, there are only two critical points remaining for three edges and hence there is no valid embedding of the gadget. Conversely, if there is a valid embedding, then one of the connector edges does not block a critical point and hence the literal that it represents is `true`. □

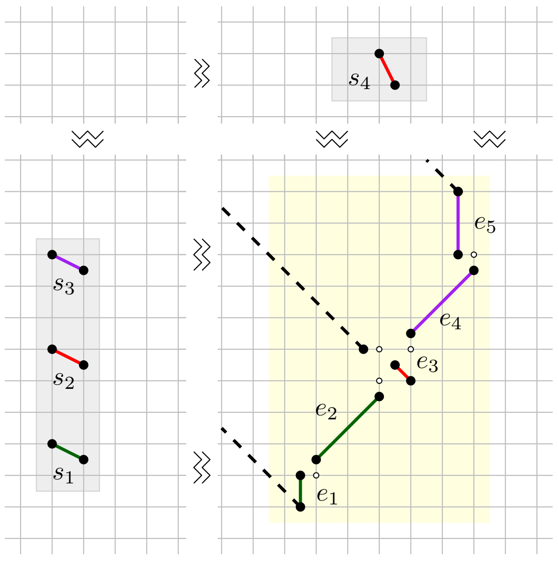


Figure 5: Sketch of a clause gadget. Critical points are marked as white disks.

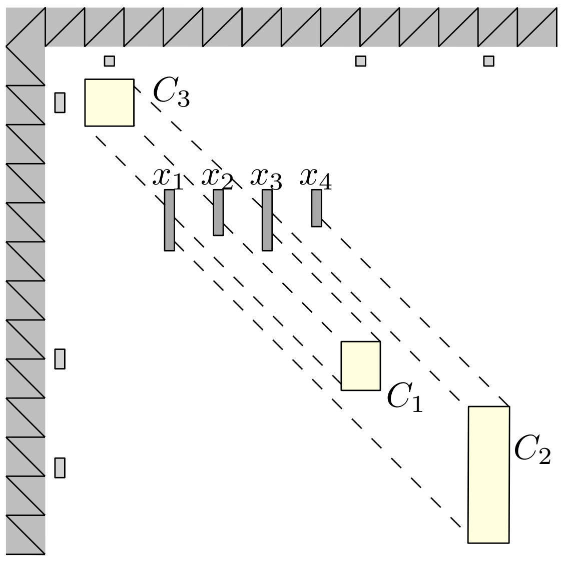


Figure 6: Sketch of the gadgets for the instance $\varphi$ from Figure 1.

The size of a clause gadget (not considering the switches) depends on the horizontal distances $\Delta_1, \Delta_2$ between the left and middle connector edge and between the middle and right connector edge, respectively. The width of a gadget is 6 and its height is $\Delta_1 + \Delta_2 - 5$.

*Gadget Placement.* The top and left part of our construction is occupied by the border gadget that induces the grid. The overall shape of the remaining construction is similar to a slanted version of the standard embedding of an instance of Planar Monotone 3-Sat as in Figure 1.

We place the individual variable gadgets horizontally aligned in the center of the drawing. Since variable gadgets do not move vertically, there is no danger of accidentally linking vertices of different variable gadgets by placing them at the same $y$-coordinates. For the clause gadgets to work as desired, we must make sure that any two connector edges to the same clause gadget have a minimum distance of seven grid cells. This can easily be achieved by spacing the variable gadgets horizontally so that connector edges of adjacent variables cannot come too close to each other.

To avoid linking between different clause gadgets or clause gadgets and variable gadgets, we place each of them in its own $x$- and $y$-interval of the grid with the negative clauses to the top left of the variables and the positive clauses to the bottom right. The switches of each clause gadget are placed at the correct positions just next to the border gadget. Figure 6 shows a sketch of the full placement of the gadgets for a Planar Monotone 3-Sat instance $\varphi$. Since the size of each gadget is polynomial in the size of the formula $\varphi$, so is the whole construction. The above construction finally yields the following theorem.

**Theorem 1.** *The 2-regular Union of Paths Schematization Problem is NP-hard.*

*Proof.* For any given Planar Monotone 3-Sat instance $\varphi$ with variables $X = \{x_1, \dots, x_n\}$ and clauses $C = \{c_1, \dots, c_m\}$ we construct and place the paths for the border, variable and clause gadgets as in Section 3.1. From Lemma 2 we know that a clause gadget has a valid embedding if and only if it contains a true literal. So the whole construction has a valid embedding if and only if every clause gadget has a valid embedding. This is the case if and only if $\varphi$ is satisfiable.

Both the size of the construction and the time to create it are polynomial in $n$ and $m$, the number of variables and clauses in $\varphi$. $\square$

## *3.2. Hardness of the Path Schematization Problem*

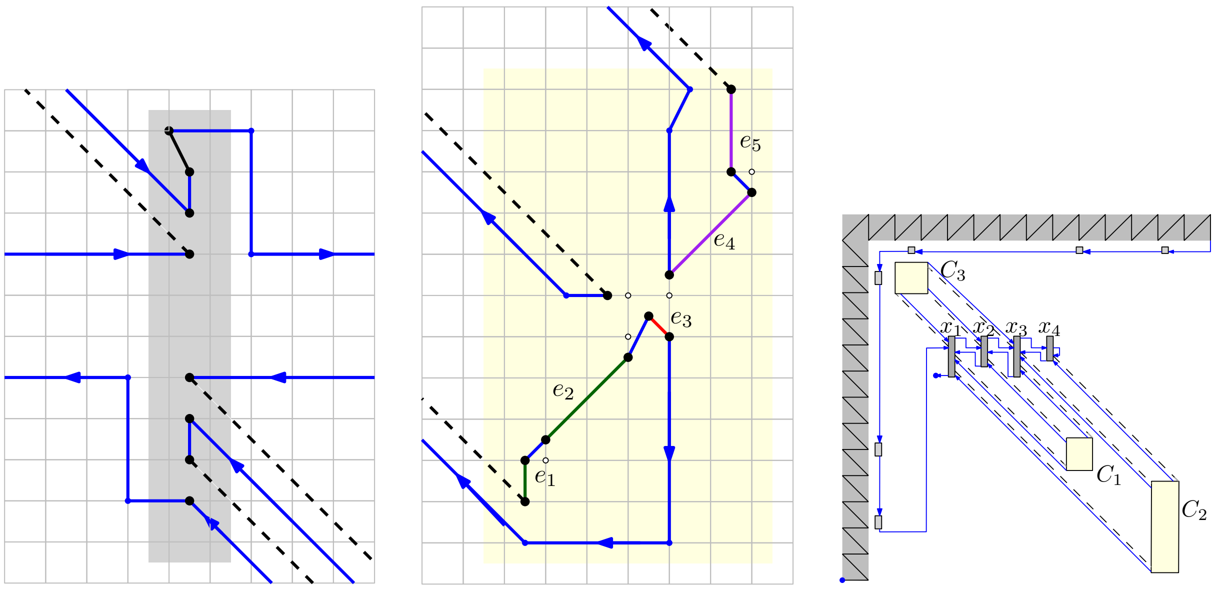
To prove that 2-regular PSP is also NP-hard we have to show that we can augment the union of paths constructed above by adding additional vertices and edges to form a single simple path that still has the property that it has a valid embedding if and only if the corresponding Planar Monotone 3-Sat formula $\varphi$ is satisfiable.

The general idea is to start the path at the lower end of the border gadget and then collect all the switches next to the border gadget. From there we enter the upper parts of the variable gadgets and walk consecutively along all the connector edges into the negative clause gadgets. Once all negative connectors have been traversed, the path continues along the positive connectors and into the positive clauses. The additional edges and vertices must be placed such that they do not interfere with any functional part of the construction.

The only major change is that we double the number of connector vertices in each variable gadget and add a parallel dummy edge for each connector edge. That way we can walk into a clause gadget along one edge and back into the variable gadget along the other edge. We also need a clear separation between the negative and positive connector vertices, i.e., the topmost positive connector vertices of all variable gadgets are assigned the same $y$-coordinate and we adjust the required spacing of the variable gadgets accordingly. Figure 7(a) shows an example of an augmented variable gadget with one positive and two negative connector edges.

The edges $e_1$–$e_3$ and $e_4, e_5$ of each clause gadget are inserted into the path in between the leftmost connector edge and its dummy edge and in between the rightmost connector edge and its dummy edge, respectively. The details are illustrated in Figure 7(b). It is clear that by this construction we obtain a single simple path $P$ that contains all the gadgets of our previous reduction. Moreover, the additional edges are embedded such that they do not interfere with the gadgets themselves. Rather they can move along with the flexible parts without occupying grid points that are otherwise used by the gadgets. We conclude:

**Lemma 3.** *The 2-regular Path Schematization Problem is NP-hard.*



(a) Variable gadget (b) Clause gadget (c) Sketch of full instance

Figure 7: Augmented gadget versions. Additional path edges are highlighted in blue.

## *3.3. Membership in NP*

To show that the Path Schematization Problem for $d = 2$ is NP-complete it remains to argue its membership in NP. For this we sketch a non-deterministic polynomial-time algorithm, which guesses a solution that can subsequently be verified in polynomial time. More precisely, we guess a combinatorial structure for an input instance which we then verify in a section step with a linear program (LP).

First, we guess for each pair of vertices their relative position. We distinguish between 16 different relative positions for a vertex pair. More specifically, for a vertex $v_i$ another vertex $v_j$ lies either in one of the eight sectors (shaded in gray) or on one of the eight half lines with origin at $v_i$ (black lines); see Figure 8. Further, to help us ensure that the resulting path is not self intersecting, we guess additionally for each pair of path edges which edge is completely contained in one of the half planes induced by the line obtained by extending the other edge. Finally, we also guess on which side of the line the other edge lies. Note that two line segments do not intersect if and only if one segment is fully contained in one of the half planes induced by the other segment's extension to a line.

To verify that the structure guessed as above induces a valid 2-schematization we need to determine whether there exists a schematization that (i) conforms to the structure, (ii) maintains the orthogonal order of the input, (iii) is 2-regular, and (iv) is not self intersecting. Checking whether all path edges have a direction in $\mathcal{C}_2$ can be done straightforwardly as it follows directly from the relative vertex positions. Next we sketch an LP that enables us to check if there exists a valid 2-schematization which conforms to the remaining requirements.

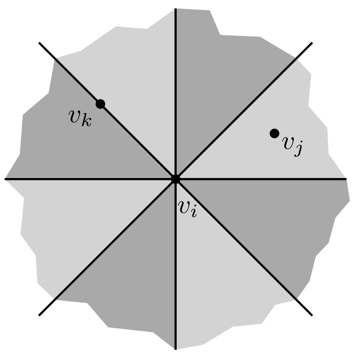


Figure 8: Possible relative positions of vertices $v_j$ and $v_k$ with respect to a vertex $v_i$. The eight sectors are shaded in gray.

For each vertex $v_i$ of the path we maintain its coordinates in variables $x_i$ and $y_i$. The first set of constraints for our LP follows directly from the orthogonal order. For all remaining constraints we need to find a way to ensure that a vertex lies to the left/on/to the right of a line. For lines that are horizontal or vertical this is trivial. For the diagonal directions we need to be more careful. To ensure that a vertex $v_j$ lies to the left/on/to the right of a line $\ell$ through $v_i$ with direction $\alpha \in \mathcal{C}_2$ we make use of the dot product. This yields that $(y_j - y_i) - \tan(\alpha)(x_j - x_i)$ is larger than 0/equal to 0/smaller than 0 if the vertex $v_j$ lies to the left/on/the right of the line $\ell$. With this observation we can easily add constraints that ensure that the relative position of each vertex pair is maintained and the path is intersection free.

We do not need an objective function since any feasible solution for our linear program induces a valid 2-schematization of the input. This is sufficient for our purposes. Note that the coefficients in some constraints are $\tan(\alpha)$ which is potentially irrational. However, for the special case $d = 2$ those numbers are rational, and hence, we can use standard linear programming algorithms (e. g, Vaidya's LP algorithm [23]) to check in polynomial time whether the LP admits a feasible solution. Hence, PSP is in NP for $d = 2$.

We can conclude that in combination with Lemma 3 the following.

**Theorem 2.** *The 2-regular Path Schematization Problem is NP-complete.*

## 4. NP-completeness of $d$-regular PSP for $d > 2$

In Section 3 we established that the 2-regular Path Schematization Problem is NP-complete. Here we show that $d$-regular PSP is NP-complete for all $d > 2$ by modifying the gadgets of our reduction for $d = 2$.

**Theorem 3.** *The $d$-regular Path Schematization Problem is NP-complete for any $d > 2$.*

*Proof.* An obvious problem for adapting the gadgets is that, due to the presence of more than one diagonal direction, the switches do not work any more for uniform grid cells. In a symmetric grid, we cannot ensure that the free vertex of a switch is always on a grid point in any valid $d$-schematization. This means that

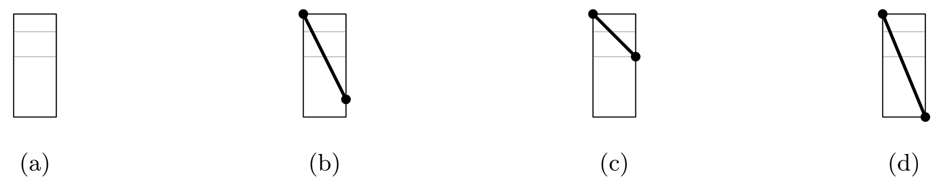


Figure 9: (a) A meta cell with its three minor cells for $d = 4$; (b) input embedding of a vertical switch; (c)+(d) the two possible output embeddings of the vertical switch.

we need to devise a different grid in order to make the switches work properly. We construct a new border gadget that induces a grid with cells of uniform width but non-uniform heights. We call the cells of this grid *minor cells* and form groups of $d-1$ vertically consecutive cells to form *meta cells*. Then all meta cells again have uniform widths and heights. For an example of a meta cell and its minor cells see Figure 9(a).

The $d-1$ different heights of the minor cells are chosen such that the segments connecting the upper left corner of a meta cell to the lower right corners of its minor cells have exactly the directions that are multiples of $(90/d)^\circ$ and lie strictly between $0^\circ$ and $90^\circ$. This restores the functionality of switches whose fixed vertex is placed on the upper left corner of a meta cell and whose free vertex is on a non-adjacent grid line of the same meta cell. See Figure 9(b) for an example of a vertical switch.

*Border Gadget.* The main purpose of the border gadget is to provide a way to control the placement of vertices on discrete positions. In Section 3 we used for this a simple gadget, which induced a regular grid. However, since for $d > 2$ there are more than one possible diagonal directions, the grid induced by the simple gadget might be highly irregular. Thus, we need to find a new border gadget that provides a similar functionality as the simple border gadget of Section 3 and which can also handle multiple diagonal directions. As for the simple case, the border gadget in this section consists also of a horizontal part $B_h$ and a vertical part $B_v$. However, the horizontal part $B_h$ of the border gadget consists of two new gadgets, the spiral- and the stairs-gadget, which we describe next.

The spiral gadget consists of a path and its main property is that in any valid $d$-schematization the first edge of this path must be drawn with the steepest non-vertical direction. This yields that the width and height of this edge have a fixed ratio. We use this later to ensure that certain vertices have the same distances.

The spiral gadget $S_k$, where $k \geq 3$, is constructed inductively. All its vertices are contained in the bounding box spanned by the vertices $v_{k-1}$ and $v_k$. Each gadget $S_k$ has an *entry vertex* $v_k$ and an *exit vertex* $u_k$ which has the same $y$-coordinate as $v_{k-1}$. The base case $S_3$ is depicted in Figure 10(a). The gadget $S_{k+1}$ is constructed by adding to $S_k$ the vertex $v_{k+1}$ and the edge $v_k v_{k+1}$ as well as a path $u_k x_{k+1} y_{k+1} z_{k+1} u_{k+1}$; the construction is illustrated in Figure 10(b). The vertex $x_{k+1}$ has the same $x$-coordinate as $u_k$ and is above it. The vertex $y_{k+1}$ has the same $y$-coordinate as $x_{k+1}$ and is to the left of $v_{k-1}$ and to the right of

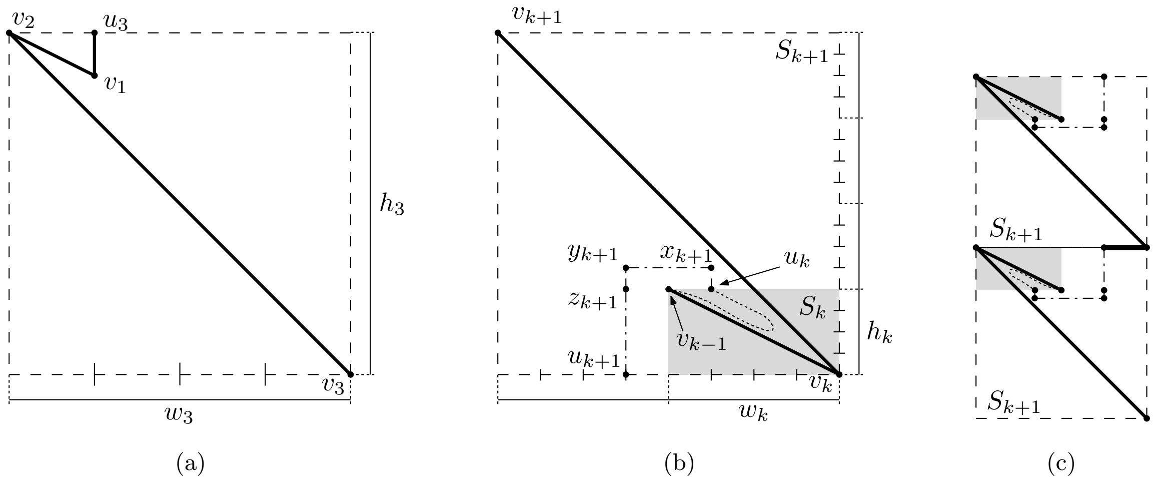


Figure 10: The spiral gadget $S_k$. (a) the spiral gadget $S_3$ and (b) the construction of $S_{k+1}$ from $S_k$ for $k \geq 3$. The gadget $S_{k+1}$ is rotated by 180° degrees to better highlight the construction. In (c) two spirals in their correct orientation placed on top of each other and connected by the bold edge.

$v_{k+1}$. The vertices $y_{k+1}$, $z_{k+1}$ and $u_{k+1}$ share the same $x$-coordinate. The vertex $y_{k+1}$ is vertically aligned with $x_{k+1}$, the vertex $z_{k+1}$ is vertically aligned with $u_k$, and the vertex $u_{k+1}$ is vertically aligned with $v_k$. Note that all vertices are contained in the bounding box spanned by $v_k$ and $v_{k+1}$. Finally, we rotate the result by 180° degrees.

We now discuss the size of a single spiral gadget. We denote the width of a spiral gadget $S_k$ by $w_k$ and its height by $h_k$. In each step of iterative construction we position the new vertex $v_{k+1}$ such that the edge $v_k v_{k+1}$ spans a box with width $2 \cdot w_k$ and height $4 \cdot h_k$. This directly induces a grid on which we can place the vertices and it implies that we require coordinates in $O(4^k)$ to describe a gadget $S_k$; see Figure 10(b). However, to ensure that we can construct a path that can leave the spiral gadget we need to increase the size of the grid by a factor of 4 both vertically and horizontally. With this increase, vertex coordinates in $O(16^k)$ are sufficient to describe a gadget $S_k$. Although this might seem problematic, please note that the size of our spiral gadget $S_d$ does only depend on $d$ which is not part of the input and thus the coordinates to describe the spiral gadget are in $O(1)$.

**Lemma 4.** *For any $k \geq 2$ and any valid d-schematization of $S_{k+1}$, the edge $v_{k+1}v_k$ is steeper than $v_k v_{k-1}$.*

*Proof.* For $k = 2$ the statement follows since $u_3$ and $v_1$ must be embedded above the edge $v_2 v_3$. Now let $k > 2$ and let $(S_{k+1}, \rho)$ be a valid $d$-schematization. For simplicity we consider the drawing that is obtained by a rotation of 180° degrees. Then the construction looks as depicted in Figure 10(b). Observe that this preserves the directions of the edges. Hence, the statement of the lemma holds if $v_{k-1}$ is embedded below the edge $v_k v_{k+1}$. Otherwise, $v_{k-1}$ is embedded above $v_k v_{k+1}$. Since $u_k$ is to the right of $v_{k-1}$, it follows that $u_k$

is also embedded above $v_kv_{k+1}$. Since $u_{k+1}$ is vertically aligned with $v_k$, it is embedded below $v_kv_{k+1}$. The path starting at $v_{k-1}$ and consisting of $S_k$, $u_k$, $x_{k+1}$, $y_{k+1}$, $z_{k+1}$ and $u_{k+1}$ is completely contained in the bounding box spanned by $v_kv_{k+1}$. On the other hand, this path connects $u_k$ with $u_{k+1}$, which lie on different sides of $v_kv_{k+1}$. This contradicts the planarity of $\rho$. □

**Lemma 5.** *For any $d \geq 3$ the spiral gadget $S_d$ has a valid d-schematization. In any valid d-schematization of $S_d$ the edge $v_dv_{d-1}$ has the steepest non-vertical direction.*

*Proof.* For any $d \geq 3$, the gadget $S_3$ has a valid $d$-schematization such that the edge $v_1v_2$ has direction $90°/d$ and $v_2v_3$ has direction $2 \cdot 90°/d$. To construct a drawing for $S_{k+1}$ we use a drawing of $S_k$ which uses the directions $90°/d, \ldots, (k-1) \cdot 90°/d$. Again, for simplicity we consider the rotated drawing as in Figure 10(b). We draw $v_{k+1}v_k$ with direction $k \cdot 90°/d$ such that $v_{k+1}$ is strictly to the left of $v_{k-1}$. The path from $u_k$ to $u_{k+1}$ is drawn orthogonally. This proves the first statement of the lemma.

We now prove the second statement. Let $d \geq 3$ and let $\rho$ be a valid $d$-schematization of $S_d$. Since $\rho$ contains a valid $d$-schematization of $S_k$ for $k = 3, \ldots, d$, it follows from Lemma 4 that $v_kv_{k-1}$ is steeper than $v_{k-1}v_{k-2}$ for $k = 3, \ldots, d$. Hence, $v_1v_2, \ldots, v_{d-1}v_d$ is a sequence of $d-1$ edges that must be embedded with increasing directions. Since $v_1v_2$ cannot be embedded horizontally (overlap with $u_3$) and $v_dv_{d-1}$ cannot be embedded vertically (overlap with $u_{k+1}$) and since there are only $d-1$ diagonal directions, the claim follows. □

We can place two copies of a spiral vertically above each other such that corresponding vertices have the same $x$-coordinates and join them by adding a horizontal edge between the last vertex of the lower copy and the first vertex of the upper copy; see Figure 10(c). The fact that the direction of the first edge $e_1$ is fixed shows that this structure is in a sense regular; both spirals have the same width and height.

The spiral provides us with a simple way of creating a construction that contains evenly spaced points. Although, at a first glance, it seems that we might be able to use the spiral gadgets directly to form a grid, this is not the case since the vertices in the interior of the gadget would influence the embedding of the remaining gadgets. Hence, we need another gadget, the *stairs gadget*, which overcomes this drawback, but is not rigid by itself.

A stairs gadget consists of $d-1$ diagonal edges placed in a zig-zag pattern from bottom to top in such a way that per gadget only two $x$-coordinates are used; see Figures 11(a) and 11(b). We wish that in any valid schematization the edges $e_i = v_iv_{i+1}$, $i = 1, \ldots, d-1$ are all embedded diagonally with increasing directions.

To ensure that the first edge cannot be embedded horizontally, we add an additional vertex $u_1$ that shares its $y$-coordinate with $v_1$ and its $x$-coordinate with $v_2$. For each $i$ with $2 \leq i \leq d-1$, we construct two spirals that are joined as described above and such that the starting point of the first spiral has the same

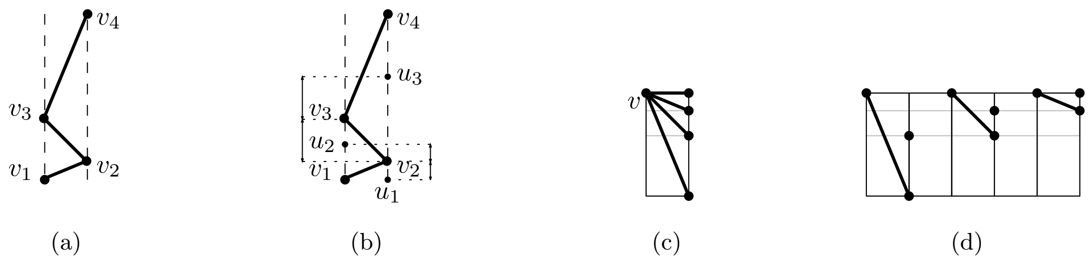


Figure 11: Substructures of the border gadget for $d = 4$. (a) The stairs gadget; (b) stairs gadget with additional vertices; (c) non-path vertical border gadget; (d) transformation of (c) into a union of paths.

$y$-coordinate as $v_i$ and its endpoint has the same $y$-coordinate as $v_{i+1}$. We then place an additional point $u_i$ vertically between $v_{i-1}$ and $v_{i+1}$ that has the same $y$-coordinate as the endpoint of the upper spiral.

The properties of the spirals imply that the vertical distance between $u_i$ and $v_i$ is the same as the vertical distance between $v_{i-1}$ and $v_i$ in any valid $d$-schematization. Since $v_{i+1}$ must be embedded strictly above $u_i$ the edge $e_{i+1}$ must be embedded steeper than $e_i$. As we have only $d-1$ diagonal directions the fact that the stairs gadget has $d-1$ edges shows that each of them is used exactly once and in increasing order along the path.

Placing $k$ identical stairs directly next to each other yields $k$ unit-width grid rows. If we link the corresponding vertices of the stairs gadgets with each other we need to place the vertices $u_i$ only in one of the stairs gadgets. Since we want to combine all gadgets to a path in the end, we use this observation and create a single stairs gadget with the additional vertices to the left of all other stairs gadgets. We can now create a path starting in this leftmost stairs gadget and then connecting the remaining $k-1$ stairs gadgets.

Now that we have established the horizontal part $B_h$ of the border gadget we can assume that all grid columns are of equal width 1. Hence the meta grid cells have height $\tan((d-1)/d \cdot 90°)$. We want to choose the heights of the minor cells such that the line segments from the upper left corner of the meta cell to all grid points on the right side of the meta cell have a direction in $\mathcal{C}_d$; see Figure 11(c). This property is necessary for constructing switches as explained later.

Unfortunately, the gadget in Figure 11(c) is not a path and hence we cannot use it to induce the required grid structure directly. Instead, we "stretch" the structure and use the concept of linking in order to yield the same result. We use $d-1$ grid columns—one for each diagonal direction—with one "empty" grid column between any two non-empty grid columns; see Figure 11(d). To induce $\ell$ grid rows we repeat this structure $\ell$ times.

The horizontal and vertical parts of the border gadget are combined by placing the vertical parts such that all vertices are placed on grid lines. A sketch of the complete border gadget for $d = 4$ is shown in Figure 12.

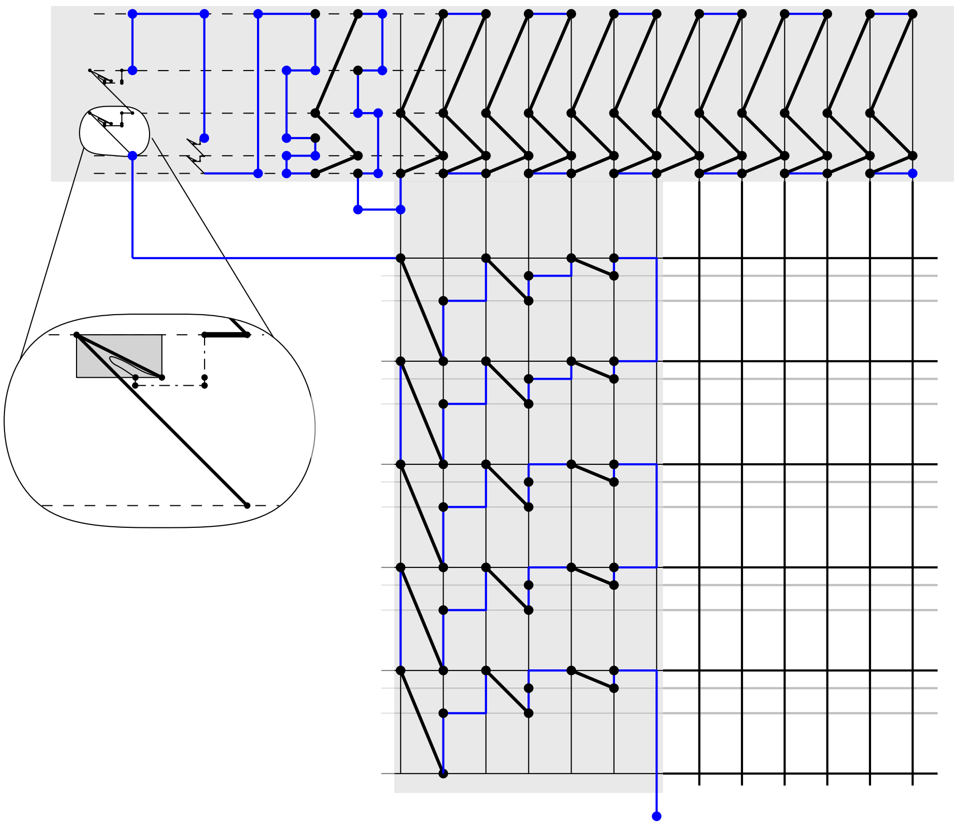


Figure 12: Sketch of the border gadget for $d = 4$.

*Switches.* Since the border gadget induces a non-regular grid, the switches must be adjusted. Each switch starts at the upper left corner of a meta cell. For a horizontal switch the free vertex is placed on the lower edge of the meta cell. For a vertical switch the free vertex is placed on the right edge of the meta cell in between two horizontal grid lines.

*Variable and Clause Gadgets.* The variable gadget is unchanged, given that all its vertices are placed on meta grid lines. The overall structure and functionality of the clause gadgets remains unchanged as well. However, we need to do some slight modifications; see Figure 13. The critical edge $e_3$ is placed in the topmost minor cell of a meta cell. Due to the nature of our border gadget and the induced grid the diagonal edge $e_2$ starts at the second-to-top minor cell of a meta cell and the diagonal edge $e_4$ starts at the bottommost minor cell of a meta call. The edge $e_1$ must have its upper vertex on the lower grid line of the second-to-top minor cell. The edge $e_5$ must have its lower vertex on the top grid line of the bottommost cell in a meta cell.

*Gadget Placement and Single Path.* The placement of the gadgets is identical as for $d = 2$. The variable gadgets are placed sufficiently spaced at the same height along the $x$-axis. The negative clauses are to their

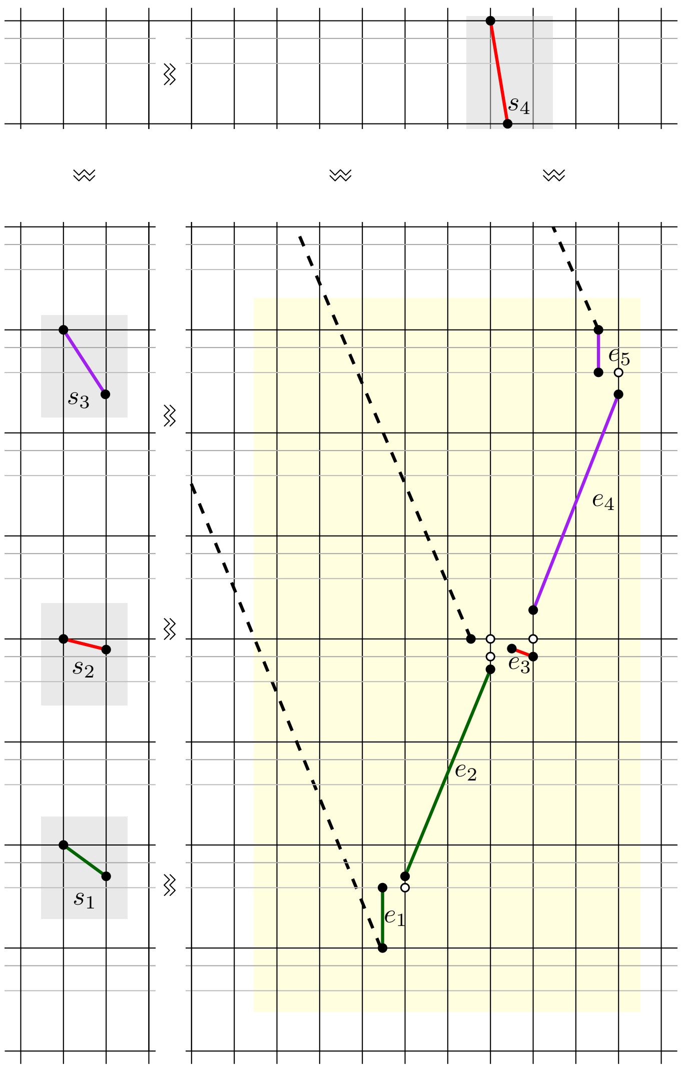


Figure 13: Sketch of a clause gadget for $d = 4$.

upper left and the positive clauses to their lower right. The border gadget is chosen large enough to induce a grid that covers all the gadgets. Combining the gadgets to a single simple path works analogously to the case $d = 2$ and the size of all gadgets is polynomial.

*Membership in NP.* It remains to argue that the Path Schematization Problem is in NP for $d > 2$. We use the same idea described in Section 3.3. First, we guess the relative vertex positions and the constraints which ensure that the path is not self-intersecting and then formulate the LP. Note that the constraints are essentially the same with the minor difference that coefficients in the constraints can be of the form $\tan(^{i}/_{(2d)} \cdot \pi)$. Those numbers might be irrational which poses a problem for standard LP algorithms. However, since $^{i}/_{(2d)}$ is a rational number and $\tan(r \cdot \pi)$ with $r \in \mathbb{Q}$ is an algebraic number [24, 25] we can use the LP algorithm by Beling [24] to test for feasibility in polynomial time. Hence, we can conclude that the Path Schematization Problem is in NP for $d > 2$. This concludes the proof of Theorem 3. $\square$

**Corollary 1.** *The $d$-regular Path Schematization Problem is NP-complete for any $d \geq 1$.*

*Proof.* From Theorem 2, Theorem 3, and the result of Brandes and Pampel [18] for $d = 1$ it follows that the $d$-regular Path Schematization Problem is NP-complete for any $d \geq 1$. $\square$

*Discussion.* We have shown that the $d$-regular Path Schematization Problem is NP-complete for $d \geq 2$. However, the gadgets in our proofs heavily rely on the concept of linking, which requires vertices to either have the same $x$- or the same $y$-coordinates. This is contrast to the proof by Brandes and Pampel [9, 18] which does not rely on this. In fact the authors argue that the proof for $d = 1$ remains valid for points in general position, i. e., no three points lie on the same line. Unfortunately, it seems to be highly non-trivial to adapt our gadgets for this case. It is unclear whether or not it is possible to construct a gadget that is similar to our border gadget in that it produces a regular grid (for $d = 2$). Hence, we cannot make a statement on the complexity of the Path Schematization Problem under the assumption that the input points are in general position. This is an interesting open question which requires additional research.

## 5. Monotone Path Schematization

In the previous sections we considered the problem of deciding for a given embedded path if it admits a $d$-regular schematization. For the practical application of generating route sketches, we are not only interested if there is such an embedding but we want to determine an embedding with minimum schematization cost, i.e., a $d$-regular schematization where a maximum number of edges have their preferred direction. Since we know from Corollary 1 that the Path Schematization Problem is NP-complete, even if we do not consider schematization cost we cannot hope for an efficient algorithm that determines a $d$-regular embedding of a given input path, unless P = NP. Hence, in this section we explore the complexity of a more restricted variant of the problem.

We show that, if we restrict the input to an axis-monotone, i.e., paths that are $x$- or $y$-monotone, embedded path, the $d$-regular path schematization problem can be solved efficiently. We devise an algorithm that solves this problem while simultaneously minimizing the schematization cost, and guaranteeing that each edge has length larger than some given minimum length. Moreover, we present a heuristic to extend this approach to general simple paths by first splitting the input path into a minimum number of axis-monotone paths, then applying the algorithm for each path separately, and finally concatenating the paths such that the resulting path is intersection-free.

We begin by stating the exact definition of the problem:

**Problem 1** (Monotone-Path Schematization Problem (MPSP)). *Given an embedded axis-monotone path $(P = (V, E), \pi)$, an integer $d \geq 1$, a minimum length $\ell_{\min}(e)$ for each edge $e \in E$, find a valid $d$-schematization $\rho$ such that:*

*(i) the schematization cost $c(\rho)$ is minimum,*

*(ii) for every edge $e \in E$ the length $|\rho(e)|$ is at least $\ell_{\min}(e)$, and*

As a practical requirement we want to find a path with minimum schematization cost *and* with minimal total path length. Note that schematization cost and total path length are two potentially conflicting optimization criteria. Primarily, we want to find a $d$-regular schematization that minimizes the schematization cost.

## *5.1. Solving the Monotone-Path Schematization Problem*

In the following we describe an efficient two-step algorithm that solves MPSP optimally for $x$-monotone paths. The algorithm can be adapted to handle $y$-monotone paths straightforwardly. In the first step the algorithm computes a valid $d$-schematization $\rho$ of the input path $(P, \pi)$ with minimum schematization cost. In the second step we adjust the embedding $\rho$ that was computed in the first step to ensure the minimum edge length requirement is met and the total path length is minimized.

### *5.1.1. Minimizing the Schematization Cost*

The goal in the first step of the algorithm is to find a valid $d$-schematization $\rho$ of $(P, \pi)$ with minimum schematization cost. To this end we argue that a given special order on the $y$-coordinates of vertices of $(P, \pi)$ induces a valid $d$-schematization of $(P, \pi)$ with minimum schematization cost among all valid $d$-schematization of $(P, \pi)$ that respects this order. We then use this insight to give a dynamic programming algorithm that computes such an order by assigning integer value between 1 and $n$ to the $y$-coordinate of every vertex of $P$ to generate the output embedding $\rho$ with minimum schematization cost. But first we explain the direction cost considered in this section.

We begin by assigning the preferred direction $\omega_{\mathcal{C}}(e) = \gamma$ to each edge $e \in P$, where $\gamma \in \mathcal{C}_d$ is the direction closest to $\alpha_\pi(e)$. We now group all edges $e = uv$ of $P$ into four categories according to their preferred direction $\omega_{\mathcal{C}}(e)$:

1. if $\omega_{\mathcal{C}}(e) = 0°$ and $y_\pi(u) \neq y_\pi(v)$, then $e$ is called horizontal edge (or *h-edge*);
2. if $y_\pi(u) = y_\pi(v)$, then $e$ is called strictly horizontal edge (or *sh-edge*);
3. if $\omega_{\mathcal{C}}(e) \neq 0°$ and $x_\pi(u) \neq x_\pi(v)$, then $e$ is called vertical edge (or *v-edge*);
4. if $x_\pi(u) = x_\pi(v)$, then $e$ is called strictly vertical edge (or *sv-edge*).

Using these categories, we define the direction cost for an edge as follows. All edges $e$ with $\omega_{\mathcal{C}}(e) = \alpha_\rho(e)$ are drawn according to their preferred direction and we assign the cost $c_\rho(e) = 0$. For all edges $e$ with $\omega_{\mathcal{C}}(e) \neq \alpha_\rho(e)$ we assign the cost $c_\rho(e) = 1$. An exception are the sh- and sv-edges, which must always be assigned their preferred direction due to the orthogonal ordering constraints. Consequently, we set $c_\rho(e) = \infty$ for any sh- or sv-edge $e$ with $\omega_{\mathcal{C}}(e) \neq \alpha_\rho(e)$.

Note that assigning each edge its preferred direction might result in the following conflict. Consider two subsequent edges $e_1, e_2$ with $\{\omega_{\mathcal{C}}(e_1), \omega_{\mathcal{C}}(e_2)\} = \{90°, 270°\}$. Assigning such preferred directions would result in an overlap of $e_1$ and $e_2$. In this case, we either set $\omega_{\mathcal{C}}(e_1)$ or $\omega_{\mathcal{C}}(e_2)$ to its next best value, depending on which edge is closer to it. This neither changes the solution nor creates new conflicts since in a plane embedding not both edges can have their preferred direction.

In the following we show that for a special kind of order, we call *$y$-order*, on the $y$-coordinates of the vertices of $(P, \pi)$ we can always find a valid $d$-schematization. Moreover, we show that for a given $y$-order we can easily construct a valid $d$-schematization with minimum schematization cost among all valid $d$-schematizations respecting this order. A $y$-order for a given path $(P, \pi)$ is an order on the $y$-coordinates of the vertices of $P$ that (i) respects the orthogonal order of $(P, \pi)$ and (ii) for every pair of vertices $v, w$ in $P$ the order tells us if the vertices share the same $y$-coordinate or if one lies strictly above the other. We are now ready to state the lemma.

**Lemma 6.** *Let $(P, \pi)$ be a path with $x$-monotone embedding, and let $\mathcal{Y}$ be a $y$-order for $(P, \pi)$. Then, we can construct a valid $d$-schematization that respects $\mathcal{Y}$ with minimum schematization cost among all valid $d$-schematizations of $(P, \pi)$ that respect $\mathcal{Y}$.*

*Proof.* Since we assume the input embedding to be $x$-monotone, the output embedding $\rho$ must be $x$-monotone, too, as it preserves the orthogonal order of $\pi$. So we can assume that $P = (v_1, \ldots, v_n)$ is ordered from left to right in both embeddings. Let $\rho'$ be any orthogonal-order preserving embedding of $\pi$. We start with the observation that in $\rho'$ every edge $e = v_i v_{i+1}$ with $\omega_{\mathcal{C}}(e) \neq 0°$ and $y_{\rho'}(v_i) \neq y_{\rho'}(v_{i+1})$ can be embedded with its preferred direction $\alpha_{\rho'}(e) = \omega_{\mathcal{C}}(e)$. This is achieved by horizontally shifting the whole embedding $\rho'$ right of $x_{\rho'}(v_{i+1})$ (including $v_{i+1}$) to the left or to the right until the direction of $e$ satisfies $\alpha_{\rho'}(e) = \omega_{\mathcal{C}}(e)$. Due to the $x$-monotonicity of $P$ no other edges are affected by this shift.

Now, to see how to obtain a valid $d$-schematization with minimum schematization cost consider the following. Assume $\rho$ is a valid $d$-schematization of $(P, \pi)$ that respects $\mathcal{Y}$. For any edge $e = v_i v_{i+1}$ in $P$ that is an h-edge (sh-edge) the direction cost induced by $\mathcal{Y}$ is 1 ($\infty$) if and only if $\mathcal{Y}$ implies $y_\rho(v_i) \neq y_\rho(v_{i+1})$; otherwise the direction cost is 0. Likewise, if $e$ is a v-edge (sv-edge) then the direction cost induced by $\mathcal{Y}$ is 1 ($\infty$) if and only if $\mathcal{Y}$ implies $y_\rho(v_i) = y_\rho(v_{i+1})$; otherwise, by the above shifting argument, we can assign $e$ its preferred direction and hence, its direction cost is 0. Obviously, there can be no valid $d$-schematization of $(P, \pi)$ that respects $\mathcal{Y}$ with less schematization cost.

It remains to prove that there exists a valid $d$-schematization $\rho$ that respects $\mathcal{Y}$. We give a constructive proof. Start by determining $y$-coordinates according to $\mathcal{Y}$. We assign from bottom to top (according to $\mathcal{Y}$) integer values between 1 and $n$ to the vertices as $y$-coordinates. Note that two vertices, that according to $\mathcal{Y}$ share the same $y$-coordinate, get assigned the same $y$-coordinate. Since we have now assigned each vertex a $y$-coordinate, and as we have seen, the $y$-order already determines for which edges we can assign their preferred direction and for which edges this is not possible, it remains to determine the $x$-coordinates $x_\rho$ of the vertices of $P$. Note that the $x$-coordinate of each vertex $v_i \in P$, $0 < i < n$ in $\rho$ depends only on the $y$-coordinate of its predecessor $v_{i-1}$ and the assigned direction $\alpha_\rho(v_i v_{i+1})$. Hence, we are now able to obtain a valid $d$-schematization $\rho$ of $(P, \pi)$ solely based on a given $y$-order $\mathcal{Y}$. □

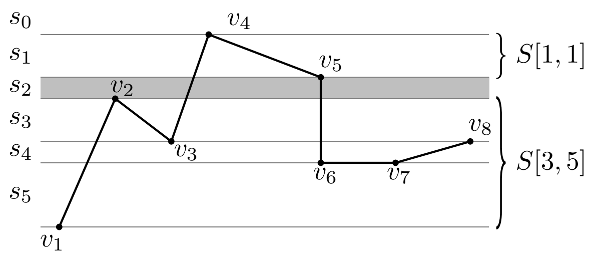
Recall that the aim for the first step of our two-step approach was to determine a valid $d$-schematization of a given $x$-monotone embedded input path $(P, \pi)$. Note that Lemma 6 implies we only need to compute a $y$-order that induces a valid $d$-schematization with minimum schematization cost to obtain such a schematization $\rho$.

In the following we give a dynamic programming algorithm that implicitly computes a $y$-order for an input path $(P, \pi)$ such that it induces a $d$-regular schematization of $(P, \pi)$ with minimum schematization cost. The algorithm assigns each vertex a $y$-coordinate between 1 and $n$. The complete valid $d$-schematization can then be obtained according to the proof of Lemma 6.

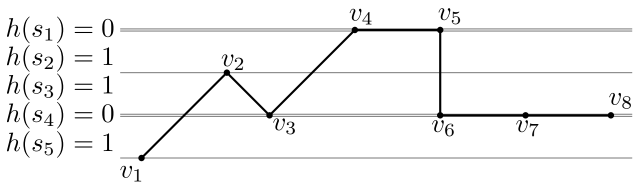
In order to assign each vertex a $y$-coordinate between 1 and $n$, we define $m \leq n-1$ closed and vertically bounded horizontal strips $s_1, \ldots, s_m$ induced by the set $\{y = y_\pi(v_i) \mid 1 \leq i \leq n\}$ of horizontal lines through the vertices of $(P, \pi)$. Let these strips be ordered from top to bottom as shown in Figure 14(a). Furthermore we define a dummy strip $s_0$ above $s_1$ that is unbounded on its upper side. We say that an edge $e = uv$ *crosses* a strip $s_i$ and conversely that $s_i$ *affects* $e$ if $\pi(u)$ and $\pi(v)$ lie on opposite sides of $s_i$. In fact, to determine the cost of an embedding $\rho$ it is enough to know for each strip whether it has a positive height or not since this induces a $y$-order on the vertices. The algorithm will assign a symbolic height $h(s_i) \in \{0, 1\}$ to each strip $s_i$, such that the resulting $y$-order of the vertices induces a valid $d$-schematization of the input path with minimum schematization cost. Note that sh-edges do not cross any strip but rather coincide with some strip boundary. Hence all sh-edges are automatically drawn horizontally and have no direction costs. We can therefore assume that there are no sh-edges in $(P, \pi)$.

Let $S[i,j] = \bigcup_{k=i}^{j} s_k$ be the union of the strips $s_i, \ldots, s_j$ and let $\mathcal{I}(i,j)$ be the subinstance of the path schematization problem containing all edges that lie completely within $S[i,j]$. Note that $\mathcal{I}(1,m)$ corresponds to the original instance $(P, \pi)$, whereas in general $\mathcal{I}(i,j)$ is not necessarily a connected path but may consist of a collection of edges. The following lemma is key to our algorithm.

**Lemma 7.** *Let $\mathcal{I}(i,j)$ be a subinstance of the monotone-path schematization problem and let $s_k \subseteq S[i,j]$ be a strip for some $i \leq k \leq j$. If we assign $h(s_k) = 1$, then $\mathcal{I}(i,j)$ decomposes into the two independent*



(a) Horizontal strips for $(P, \pi)$



(b) Strip height assignment and induced output path $(P, \rho)$.

Figure 14: Example of (a) an $x$-monotone embedded input path $(P, \pi)$ and (b) a 2-regular schematization $(P, \rho)$.

*subinstances $\mathcal{I}(i, k-1)$ and $\mathcal{I}(k+1, j)$. The direction costs of all edges affected by $s_k$ are determined by setting $h(s_k) = 1$.*

*Proof.* We first show that the cost of any edge $e = uv$ that crosses $s_k$ is determined by setting $h(s_k) = 1$. Since $u$ and $v$ lie on opposite sides of $s_k$ we know that $y_\rho(u) \neq y_\rho(v)$. So if $e$ is a v- or sv-edge, it can be drawn with its preferred direction and $c_\rho(e) = 0$ regardless of the height of any other strip crossed by $e$. Conversely, if $e$ is an h-edge it is impossible to draw $e$ horizontally regardless of the height of any other strip crossed by $e$ and $c_\rho(e) = 1$. Recall that sh-edges do not cross any strips. Assume that $k = 2$ in Figure 14(a) and we set $h(s_2) = 1$; then edges $v_3v_4$ and $v_5v_6$ cross strip $s_2$ and none of them can be drawn horizontally.

The remaining edges of $\mathcal{I}(i, j)$ do not cross $s_k$ and are either completely contained in $S[i, k-1]$ or in $S[k+1, j]$. Since the costs of all edges affected by $s_k$ are independent of the heights of the remaining strips in $S[i, j] \setminus \{s_k\}$, we can solve the two subinstances $\mathcal{I}(i, k-1)$ and $\mathcal{I}(k+1, j)$ independently; see Figure 14(a). □

*Our Algorithm.* We are now ready to describe our algorithm for assigning symbolic heights to all strips $s_1, \ldots, s_m$, and thereby determining integer $y$-coordinates for the vertices of $P$, such that the induced embedding $\rho$ has minimum schematization cost. The main idea is to recursively compute an optimal solution for each instance $\mathcal{I}(1, i)$ by finding the best $k \leq i$ such that $h(s_k) = 1$ and $h(s_j) = 0$ for $j = k+1, \ldots, i$. By using dynamic programming we can compute an optimal solution for $\mathcal{I}(1, m) = (P, \pi)$ in $O(n^2)$ time.

Let $C(k, i)$ for $1 \leq k \leq i$ denote the schematization cost of all edges in the instance $\mathcal{I}(1, i)$ that either cross $s_k$ or have both endpoints in $S[k+1, i]$ if we set $h(s_k) = 1$ and $h(s_j) = 0$ for $j = k+1, \ldots, i$. Let $C(0, i)$ denote the schematization cost of all edges in the instance $\mathcal{I}(1, i)$ if $h(s_j) = 0$ for all $j = 1, \ldots, i$. We use an array $T$ of size $m + 2$ to store the minimum schematization cost $T[i]$ of the instance $\mathcal{I}(1, i)$. Then $T[i]$ is recursively defined as follows

$$T[i] = \begin{cases} \min_{0 \leq k \leq i}(T[k-1] + C(k, i)) & \text{if } 1 \leq i \leq m \\ 0 & \text{if } i = 0 \text{ or } i = -1. \end{cases} \tag{1}$$

Together with $T[i]$ we store the index $k$ that achieves the minimum value in the recursive definition of $T[i]$ as $\kappa[i] = k$. This allows us to compute the actual strip heights using backtracking. Note that $T[m] < \infty$

since, e. g., the solution that assigns height 1 to every strip induces cost 0 for all sv-edges (recall that we can assume that there are no sh-edges in $(P, \pi)$ and hence assigning height 1 to every strip is a valid solution). Obviously, we need $O(m)$ time to compute each entry in $T$ assuming that the schematization cost $C(k, i)$ is available in $O(1)$ time. This yields a total running time of $O(m^2)$.

The next step is to precompute the schematization cost $C(k, i)$ for any $0 \le k \le i \le m$ such that it can be accessed in $O(1)$ time. Two sources contribute to the schematization cost. The first source is the schematization cost of all edges that are affected by $s_k$. As observed in Lemma 7, all v- and sv-edges crossing $s_k$ have no cost. On the other hand, every h-edge that crosses $s_k$ has cost 1. So we need to count all h-edges in $\mathcal{I}(1, i)$ that cross $s_k$. The second source is the cost of all edges that are completely contained in $S[k+1, i]$. Since $h(s_{k+1}) = \ldots = h(s_i) = 0$, we observe that any h-edge in $S[k+1, i]$ is drawn horizontally at no cost. In contrast, no v- or sv-edge $e$ in $S[k+1, i]$ attains its preferred direction $\omega_{\mathcal{C}}(e) \neq 0°$. Hence every v-edge in $S[k+1, i]$ has cost 1 and every sv-edge has cost $\infty$. So we need to check whether there is an sv-edge contained in $S[k+1, i]$ and if this is not the case count all v-edges contained in $S[k+1, i]$.

In order to efficiently compute the values $C(k, i)$ we assign to each strip $s_i$ three sets of edges. Let $H(i)$ (resp. $V(i)$ or $SV(i)$) be the set of all h-edges (resp. v-edges or sv-edges) whose lower endpoint lies on the lower boundary of $s_i$. We can compute $H(i)$, $V(i)$, and $SV(i)$ in $O(n)$ time for all strips $s_i$. Then, for $k \le i$, the number of h-edges in $H(i)$ that cross $s_k$ is denoted by $\sigma_H(k, i)$ and the number of v-edges in $V(i)$ that do *not* cross $s_k$ is denoted by $\sigma_V(k, i)$. Finally, let $\sigma_{SV}(k, i)$ be the number of sv-edges in $SV(i)$ that do *not* cross $s_k$.

This allows us to compute the values $C(k, i)$, $0 \le k \le i \le m$, recursively as follows

$$C(k,i) = \begin{cases} \infty & \text{if } \sigma_{SV}(k,i) \ge 1 \\ C(k, i-1) + \sigma_H(k,i) + \sigma_V(k,i) & \text{if } k \le i-1 \\ \sigma_H(k,k) & \text{if } k = i. \end{cases} \tag{2}$$

Since each edge appears in exactly one of the sets $H(i)$, $V(i)$, or $SV(i)$ for some $i$, it is counted towards at most $n-1$ values $\sigma_H(\cdot, i)$, $\sigma_V(\cdot, i)$, or $\sigma_{SV}(\cdot, i)$, respectively. Thus, for computing all the values $C(i, k)$, we need $O(n^2)$ time and require a table size of $O(n^2)$. However, the space requirement can be reduced to $O(n)$ as follows. We compute and store the values $T[i]$ in the order $i = 1, \ldots, n-1$. For computing the entry $T[i]$ we use only the values $C(\cdot, i)$. To compute the next entry $T[i+1]$, we first compute the values $C(\cdot, i+1)$ from $C(\cdot, i)$ and then discard all $C(\cdot, i)$. This reduces the required space to $O(n)$. We can summarize the results in the following lemma.

**Lemma 8.** *There exists an algorithm that computes the array $T$ of schematization costs in $O(n^2)$ time and $O(n)$ space.*

It remains to determine the strip height assignments corresponding to the schematization cost in $T[m]$

and show the optimality of that solution. We initialize all heights $h(s_i) = 0$ for $i = 1, \dots, m$. Recall that $\kappa[i]$ equals the index $k$ that minimized the value $T[i]$ in (1). To find all strips with height 1, we initially set $j = m$. If $\kappa[j] = 0$ we stop; otherwise we assign $h(s_{\kappa[j]}) = 1$, update $j = \kappa[j] - 1$, and continue with the next index $\kappa[j]$ until we hit $\kappa[j] = 0$ for some $j$ encountered in this process. Let $\rho$ be the $d$-regular schematization of $(P, \pi)$ induced by this strip height assignment; see Figure 14(b). We now show the optimality of $\rho$ in terms of the schematization cost.

**Theorem 4.** *Given an $x$-monotone embedded path $(P, \pi)$ and a set $\mathcal{C}$ of $d$-regular edge directions, our algorithm computes a a valid $d$-schematization $\rho$ of $(P, \pi)$ with minimum schematization cost $c(\rho)$.*

*Proof.* As we have already shown in Lemma 6, a given $y$-order for the vertices of an input path $(P, \pi)$ induces a valid $d$-schematization $\rho$ and with minimum schematization cost among all valid $d$-schematizations respecting the $y$-order. Since the input path is $x$-monotone and by construction there are no two adjacent edges with preferred directions $90°$ and $270°$ the embedding $\rho$ is plane. The embedding $\rho$ also preserves the orthogonal order of $\pi$ since $\rho$ does not alter the $x$- and $y$-ordering of the vertices of $P$; see Figure 14(b).

We show that $\rho$ has minimum schematization cost by structural induction. For an instance with a single strip $s$ there are only two possible solutions of which our algorithm chooses the better one. The induction hypothesis is that our algorithm finds an optimal solution for any instance with at most $m$ strips. Consider an instance with $m + 1$ strips and let $\rho'$ be any optimal $d$-regular schematization for this instance. If all strips $s$ in $\rho'$ have height $h(s) = 0$, then by the recurrence relation (1) it holds that $c(\rho) = T[m + 1] \leq C(0, m + 1) = c(\rho')$. Otherwise, let $k$ be the largest index for which $h(s_k) = 1$, in $\rho'$. When computing $T[m+1]$ our algorithm also considers the case where $s_k$ is the bottommost strip of height 1, which has a cost of $T[k - 1] + C(k, m + 1)$. If $h(s_k) = 1$ we can split the instance into two independent subinstances to both sides of $s_k$ by Lemma 7. The schematization cost $C(k, m + 1)$ contains the cost for all edges that cross $s_k$ and this cost is obviously the same as in $\rho'$ since $h(s_k) = 1$ in both embeddings. Furthermore, $C(k, m + 1)$ contains the cost of all edges in the subinstance below $s_k$, for which we have by definition $h(s_{k+1}) = \ldots = h(s_{m+1}) = 0$. Since $k$ is the largest index with $h(s_k) = 1$ in $\rho'$ this is also exactly the same cost that this subinstance has in $\rho'$. Finally, the independent subinstance above $s_k$ has at most $m$ strips and hence $T[k - 1]$ is the minimum cost for this subinstance by induction. It follows that $c(\rho) = T[m + 1] \leq T[k - 1] + C(k, m + 1) \leq c(\rho')$. This concludes the proof. □

*Alternative Costs.* In this section we have assumed that the cost for an h-edge (a v-edge) is 1 if it is not embedded with its preferred direction. However, our dynamic programming allows us to have different types of cost, even edge specific cost. Recall that the schematization cost for $C(k, i - 1)$ depends on $\sigma_H(k, i)$ and $\sigma_V(k, i)$ where $\sigma_H(k, i)$ is the number of h-edges in $H(i)$ that cross $s_k$, and $\sigma_V(k, i)$ is the number of v-edges in $V(i)$ that do not cross $s_k$. The sum of the numbers $\sigma_H(k, i)$ and $\sigma_V(k, i)$ is the cost of setting the strip $s_k$

to have height 1 and setting all strips below $s_k$ to have height 0. To enable edge specific cost we can do the following. First, we compute, as before, the sets $H(i)$ and $V(i)$. But when we determine the values for $\sigma_H(\cdot, i)$ and $\sigma_V(\cdot, i)$ we do not simply count the edges in $H(i)$ and $V(i)$, respectively, but we set $\sigma_H(\cdot, i)$ ($\sigma_V(\cdot, i)$) to be the sum of the weights of the edges in $H(i)$ ($V(i)$). Since this is done in a preprocessing step and requires only $O(n)$ time the total time complexity of the algorithm does not change.

In Theorem 4 we have shown that our algorithm works for $x$-monotone paths, and a generalization to axis-monotone paths is trivial. Further, we have argued that we can have edge-specific cost as schematization cost. We conclude this subsection with the following corollary.

**Corollary 2.** *The monotone path schematization problem (Problem 1) for an axis-monotone input path of length $n$ can be solved by an $O(n^2)$-time algorithm that computes a valid $d$-schematization $\rho$ of minimum schematization cost.*

### *5.1.2. Minimizing the Path Length*

In the first step of our algorithm we obtained a valid $d$-schematization $\rho$ of $(P, \pi)$ with minimum schematization cost in $O(n^2)$ time. However, this does not account for the edge lengths induced by $\rho$. So in the second step, we adjust $\rho$ such that the total path length is minimized and $|\rho(e)| \geq \ell_{\min}(e)$ for all $e \in P$. We make sure, however, that the orthogonal order and all directions $\alpha_\rho(e)$—and thus also the schematization cost $c(\rho)$—remain unchanged.

Note that we can immediately assign the minimum length $\ell_{\min}(e)$ to every horizontal edge $e$ in the input $(P, \rho)$ by horizontally shifting the subpaths on both sides of $e$. For any non-horizontal edge $e = uv$ the length $|\rho(e)|$ depends only on the vertical distance $\Delta_y(e) = |y_\rho(u) - y_\rho(v)|$ of its endpoints and the direction $\alpha_\rho(e)$. In fact, $|\rho(e)| = \Delta_y(e)/\sin(\alpha_\rho(e))$. So in order to minimize the path length we need to find $y$-coordinates for all strip boundaries such that $\sum_{e \in P} |\rho(e)|$ is minimized. These $y$-coordinates together with the given directions for all edges $e \in P$ (as computed in the previous step) induce the corresponding $x$-coordinates of all vertices of $P$.

For each strip $s_i$ ($i = 0, \ldots, m$) let $y_i$ denote the $y$-coordinate of its lower boundary. For every edge $e \in P$ let $t(e)$ and $b(e)$ denote the index of the top- and bottommost strip, respectively, that is crossed by $e$. Then $\Delta_y(e) = y_{t(e)-1} - y_{b(e)}$. We propose the following linear program (LP) to minimize the path length of a given $\mathcal{C}$-oriented embedded path $(P, \rho)$.

$$\begin{array}{llcll}
\text{Minimize} & \displaystyle\sum_{e \in P,\, \alpha_\rho(e) \neq 0^\circ} \left[\frac{1}{\sin \alpha_\rho(e)} \cdot (y_{t(e)-1} - y_{b(e)})\right] & & & \\
\text{subject to} & y_{t(e)-1} - y_{b(e)} & \geq & \sin \alpha_\rho(e) \cdot \ell_{\min}(e) & \forall e \in P,\ \alpha_\rho(e) \neq 0^\circ \\
& y_{i-1} - y_i & > & 0 & \forall s_i \text{ with } h(s_i) = 1 \\
& y_{i-1} - y_i & = & 0 & \forall s_i \text{ with } h(s_i) = 0
\end{array}$$

We assign to all vertices their corresponding $y$-coordinates from the solution of the LP. In a left-to-right pass over $P$ we compute the correct $x$-coordinates of each vertex $v_i$ from the vertical distance to its predecessor vertex $v_{i-1}$ and the direction $\alpha_\rho(v_{i-1}v_i)$. This yields a modified embedding $\rho'$ that satisfies all our requirements: the path length is minimized; the orthogonal order is preserved due to the $x$-monotonicity of $P$ and the constraints in the LP to maintain the $y$-order; by construction the directions of all edges are the same in $\rho$ and $\rho'$; no edge $e$ is shorter than its minimum length $\ell_{\min}(e)$. Hence, $\rho'$ solves Problem 1 and together with Lemma 8 and Theorem 4 we obtain

Linear programs can be solved efficiently in $O(n^{2.5}L)$ time using a method of Vaidya [23], where $n$ is the length of the path $P$ and $L$ is the number of input bits. Unfortunately, our linear program potentially contains irrational numbers as constant factors and standard LP algorithms cannot efficiently solve such linear programs. One might be tempted to use the linear program algorithm by Beling [24] since all coefficients are either rational or algebraic numbers [25, 26] which at first glance implies a polynomial running time. However, the algorithm has a running time that is polynomial (among other things) in the degree of the algebraic numbers which, in our case, is in $O(d)$. This, however, means that since the space required for representing $d$ is $O(\log d)$ bits, the algorithm requires exponential time in the input size. For practical applications we can simply round those irrational values to an appropriate precision. However, then we cannot make any claims with respect to the optimality of the resulting solution.

## *5.2. Extension to General Simple Paths*

In the previous section, we showed how to schematize a monotone path. Unfortunately, some routes in road networks are not axis-monotone, however, they can be decomposed into a usually small number of axis-monotone subpaths. So, we propose the following three-step heuristic to schematize general simple paths, which we call the simple path schematization (SPS) algorithm: We first split the input path $(P, \pi)$ into a minimum number of axis-monotone subpaths $(P_i, \pi_i)$, where $\pi_i$ equals $\pi$ restricted to the subpath $P_i$. We embed each $(P_i, \pi_i)$ separately according to Section 5.1. Then, we concatenate the subpaths by adding at most three *path-link* edges between the subpaths such that the resulting path $(P', \rho)$ is a simple $\mathcal{C}$-oriented path. Note that this heuristic does not guarantee to preserve the orthogonal order between vertex pairs of different subpaths.

Splitting an embedded simple path $P = (v_1, \dots, v_n)$ into the minimal number $k$ of subpaths $P_i, 1 \leq i \leq k$ with the property that each $P_i$ is an axis-monotone path can be done in a straightforward greedy fashion, starting from $v_1$. We traverse $P$ until we find the last vertices $v'$ and $v''$ that are not violating the $x$- and $y$-monotonicity, respectively. If $v'$ appears later than $v''$ on $P$, we set $P_1 = (v_1, \dots, v')$, otherwise $P_1 = (v_1, \dots, v'')$. We continue this procedure until we reach the end of $P$. This algorithm runs in $O(n)$ time and returns the minimal number $k$ of axis-monotone subpaths, indicated by Proposition 1.

**Proposition 1.** *The greedy path splitting algorithm divides the input path $P$ into a minimal number of axis-monotone subpaths in linear time.*

*Proof.* Assume $(L_1, \ldots, L_\ell)$ is an optimal solution that divides $P$ into $\ell$ $x$- or $y$-monotone subpaths, and let $(P_1, \ldots, P_k)$ be the solution obtained by the above algorithm. Observe that due to the greedy approach the following two statements hold: (i) the path $L_i$ for the smallest $i$ such that $L_i$ differs from $P_i$ contains fewer vertices than $P_i$; (ii) there cannot be any path $L_m$ that fully contains some path $P_j = (v_p, \ldots, v_q)$ and also $v_{q+1} \in L_m$. This implies that if there is a path $L_m$ that contains $P_j$ they both share the same last vertex. Combined, we get that there is no sequence of paths $L_1, \ldots, L_i$ containing more vertices than the sequence $P_1, \ldots, P_i$ and hence $k \leq \ell$. □

After splitting the input path, we schematize each subpath $(P_i, \pi_i)$ according to the approach described in Section 5.1. We obtain a $d$-regular schematization $\rho_i$ with minimum schematization cost and minimum path length for each $(P_i, \pi_i)$. For concatenating these subpaths, we must solve the following problem.

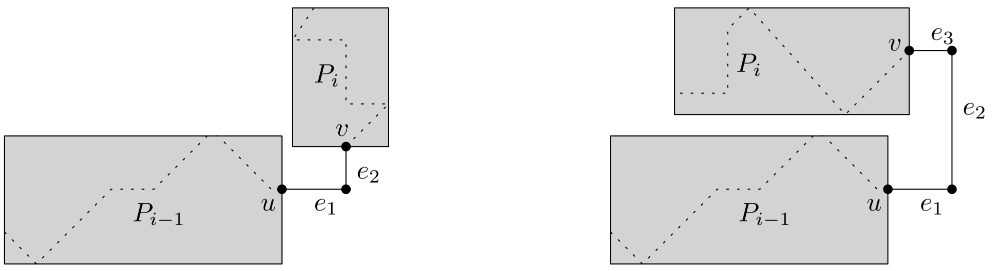
**Problem 2.** *Given a sequence of $k$ embedded axis-monotone paths $(P_i, \rho_i)$ with $1 \leq i \leq k$, find an embedding $\rho$ of $P' = P_1 \oplus \cdots \oplus P_k$, where $\oplus$ denotes the concatenation of paths, such that*

*(i) for each subpath $(P_i, \rho_i)$, the embedding $\rho$ restricted to $P_i$ is a translation of $\rho_i$ and*

*(ii) $(P', \rho)$ is a simple $d$-regular path.*

Our approach is based on iteratively embedding the subpaths $P_1, \ldots, P_k$. We ensure that in each iteration $i$ the embedding of $P_1 \oplus \cdots \oplus P_i$ remains conflict-free, i. e., it has no self-intersections. We achieve this by adding up to three new *path-link edges* between any two adjacent subpaths $P_i$ and $P_{i+1}$. For each $1 \leq i \leq k$ let $B_i$ denote the bounding box of $(P_i, \rho_i)$. We show how to construct an embedding $\rho$ of $P'$ such that for any $i \neq j$ we have $B_i \cap B_j = \emptyset$. Consequently, since each individual $(P_i, \rho_i)$ is conflict-free, $(P', \rho)$ is conflict-free as well. A key operation of the algorithm is *shifting* a subpath $P_i$ (or equivalently a bounding box $B_i$) by an offset $\Delta = (\Delta_x, \Delta_y) \in \mathbb{R}^2$. This is done by defining the lower left corner of each bounding box $B_i$ as its origin $o_i$ and storing the coordinates of $P_i$ relative to $o_i$, i. e., $\rho(v) = o_i + \rho_i(v)$. Note that shifting preserves all local properties of $(P_i, \rho_i)$, i. e., the orthogonal order as well as edge lengths and orientations.

Each iteration of our algorithm consists of two steps. First, we *attach* the subpath $P_i$ to its predecessor $P_{i-1}$. To this end, we initially place $(P_i, \rho_i)$ such that the last vertex $u$ of $P_{i-1}$ and the first vertex $v$ of $P_i$ coincide. Then we add either two path-link edges (if the monotonicity directions of $P_{i-1}$ and $P_i$ are orthogonal) or three path-link edges (if $P_{i-1}$ runs in the opposite direction of $P_i$) between $u$ and $v$ and shift $B_i$ by finding appropriate lengths for the new edges such that $B_{i-1} \cap B_i = \emptyset$. Paths $P_{i-1}$ and $P_i$ are now conflict-free, but there may still exist conflicts between $P_i$ and paths $P_j (j < i - 1)$. These are resolved in a second step that, roughly speaking, "inflates" $B_i$ starting at $v$ until it has reached its original size. Any



(a) Paths with orthogonal directions (b) Oppositely directed paths

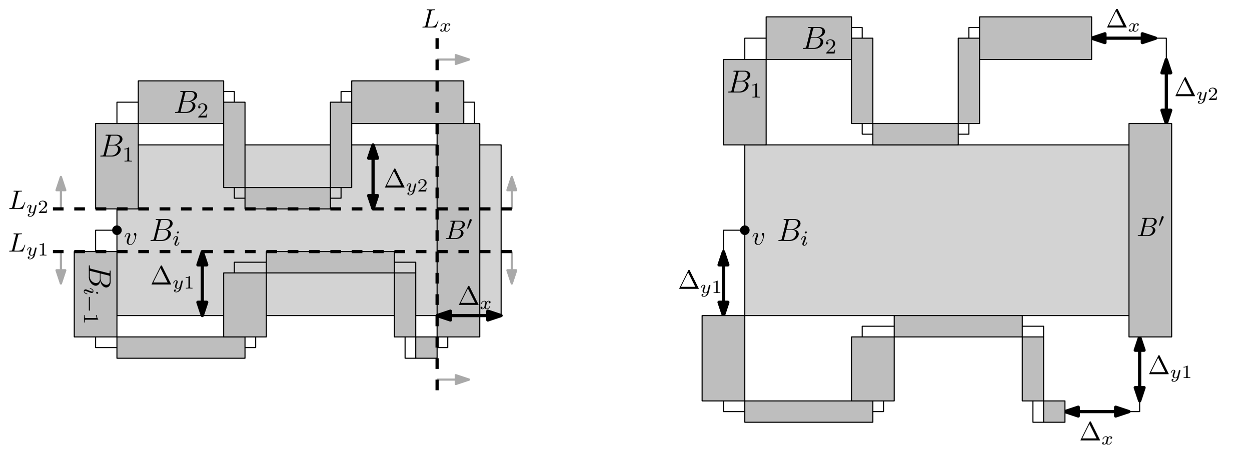
Figure 15: Two examples for attaching $P_i$ to $P_{i-1}$ by inserting path-link edges.

conflicting bounding boxes are "pushed" away from $B_i$ by stretching some of the path-link edges. In the following, we explain our procedures for attaching a subpath and resolving conflicts in more detail.

*Attaching a Subpath.* Without loss of generality, we restrict ourselves to the case that $P_{i-1}$ is an $x$-monotone path from left to right. Let $u$ be the last vertex of $P_{i-1}$ and $v$ be the first vertex of $P_i$. If $P_i$ is $y$-monotone we add a horizontal edge $e_1 = uu'$ with $\alpha_\rho(e_1) = 0°$ connecting $u$ to a new vertex $u'$. Then we also add a vertical edge $e_2 = u'v$ with $\alpha_\rho(e_1) = 90°$ if $P_i$ is upward directed and $\alpha_\rho(e_1) = 270°$ if it is a downward path. Otherwise, if $P_i$ is $x$-monotone from right to left, we add two vertices $u'$ and $u''$ and three path-link edges $e_1 = uu'$, $e_2 = u'u''$, and $e_3 = u''v$ with $\alpha_\rho(e_1) = 0°$, $\alpha_\rho(e_2) = 90°$ if $P_i$ is above $P_{i-1}$ in $\pi$ or $\alpha_\rho(e_2) = 270°$ otherwise, and $\alpha_\rho(e_3) = 180°$. Note that technically we treat each path-link edge as having its own bounding box with zero width or height. It remains to set the lengths of the path-link edges such that $B_i \cap B_{i-1} = \emptyset$ by computing the vertical and horizontal overlap of $B_{i-1}$ and $B_i$. Figure 15 illustrates both situations. Setting the lengths of the path-link edges leaves us with some degree of freedom. Consider the example depicted in Figure 15(a). To resolve the conflict we could set the length of $e_2$ to 0 and the length of $e_1$ to the appropriate length. Since in general we want to add as few edges as possible we propose to set the length of the path link edges to 0 whenever possible. Note that this is not always possible. Then, we compute the minimum lengths of $e_1$ and $e_2$ that is necessary to resolve the conflict. The edge with the smaller minimum length gets assigned this length, while the other edge length is set to a minimum value. The case depicted in Figure 15(b) can be handled similarly.

*Resolving Conflicts.* After adding $P_i$ we have $B_{i-1} \cap B_i = \emptyset$. However, there may still exist conflicts with any previously placed box $B_j, 1 \leq j < i-1$. In order to free up the space required to actually place $B_i$ without overlapping any other bounding box, we push away all conflicting boxes in three steps. For illustration, let $P_i$ be $x$-monotone from left to right, and let $v$ be the first vertex of $P_i$.

Each bounding box $B$ is defined by its lower left corner $ll(B) = (ll_x(B), ll_y(B))$ and its upper right corner $ur(B) = (ur_x(B), ur_y(B))$. In the first step we identify the leftmost box $B'$ (if any) that is intersected by a line segment that extends from $\rho(v)$ to the right with length equal to the width of $B_i$. For this box $B'$ we



(a) Before resolving the conflict (b) After resolving the conflict

Figure 16: Example for iteratively resolving conflicts induced by attaching $P_i$. First, we shift everything right of $L_x$ to the right by $\Delta_x$. Then, we shift everything below $L_{y1}$ by $\Delta_{y1}$ downward, and finally, we shift everything above $L_{y2}$ upward.

have $ll_y(B') \leq y_\rho(v) \leq ur_y(B')$ and $ll_x(B_i) \leq ll_x(B') \leq ur_x(B_i)$. If there is such a $B'$ let the offset $\Delta_x$ be $\Delta_x = ur_x(B_i) - ll_x(B')$.

Now we shift *all* bounding boxes $B$ that lie completely to the right of $ll_x(B')$ to the right by $\Delta_x$. All horizontal path-link edges (which are also considered bounding boxes by themselves) that connect a shifted with a non-shifted path are stretched by $\Delta_x$ to keep the two paths connected. Note that there is always a horizontal path-link edge between any two subsequent paths. Since $P_i$ is an $x$-monotone path from left to right, the bounding Box $B_i$ is currently a horizontal line segment.

Next, we inflate $B_i$ downwards: we first determine the topmost conflicting box $B''$ (if any) below a horizontal line through $\rho(v)$, i. e., a box $B''$ whose $x$-range intersects the $x$-range of $B_i$ and for which $ll_y(B_i) \leq ur_y(B'') \leq y_\rho(v)$. If we find such a $B''$ we define the vertical offset $\Delta_{y1} = ur_y(B'') - ll_y(B_i)$. We shift all bounding boxes $B$ that lie completely below $ur_y(B'')$ downwards by $\Delta_{y1}$. All vertical path-link edges that connect a shifted with a non-shifted box are stretched by $\Delta_{y1}$ in order to keep the two boxes connected. Again, there is always a vertical path-link edge between any two subsequent paths. Finally, we inflate $B_i$ upwards, which is analogous to the downward inflation. Figure 16 shows an example.

The approach explained above first inflates the box to the right, then downwards, and finally upwards. Of course, we could use any strategy of inflating the box. In fact, since different strategies yield different results, we could test a fixed number of strategies and keep the result that minimizes the overall increase in edge lengths.

In the following, we show that our algorithm indeed solves Problem 2.

**Lemma 9.** *Our algorithm computes a conflict-free embedding $\rho$ of $P' = P_1 \oplus \cdots \oplus P_k$.*

*Proof.* We prove the theorem by induction. By definition, the first path $P_1$ has a conflict-free embedding. Now, assume that $P_1 \oplus \cdots \oplus P_{i-1}$ is already embedded conflict-free. We attach $P_i$ to $P_{i-1}$, such that $B_i \cap B_{i-1} = \emptyset$ holds. Thus, $P_i$ and $P_{i-1}$ do not have a conflict. Next, we shift all $B_j$ with $j < i$ such that $B_i \cap B_j = \emptyset$ in the end.

What remains to be shown is that our shifting-operation does not create new conflicts between any two boxes $B_j$ and $B_{j'}$ with $j, j' < i$. Consider, e. g., a shift to the right and assume that after the shift $B_j$ and $B_{j'}$ intersect, while they did not intersect before the shift. Clearly, if none of the boxes has been moved, they cannot intersect. So either the shift moved both boxes by the same offset $\Delta_x$, or it moved only the rightmost of the two boxes to the right by $\Delta_x$. There is no vertical movement. So in both cases the horizontal distance of the boxes does not decrease and it is impossible that $B_j$ and $B_{j'}$ intersect after the shift. The same observation holds for the vertical shifts. □

**Theorem 5.** *Our algorithm computes a solution $(P', \rho)$ to Problem 2 by adding at most $3(k-1)$ path-link edges to $P$, where $k$ is the number of given embedded axis-monotone subpath. It can be implemented with a running time of $O(k^2 + n)$.*

*Proof.* The correctness of the algorithm follows from Lemma 9 and due to the fact that the embeddings $(P_i, \rho_i)$ within the bounding boxes $B_i$ remain unchanged. Clearly, we add at most three edges between any two consecutive paths, which shows the bound on the number of path-link edges.

It remains to show the running time of $O(k^2 + n)$. For the position of each bounding box we maintain the coordinates of its lower left and upper right corners. In each iteration, attaching a new subpath requires constant time since it depends only on the overlap of the new bounding box and its immediate predecessor. Resolving all conflicts requires $O(k)$ time per iteration. When adding path $P_i$ we need to check for each box $B_j$ $(j < i - 1)$ and the boxes of their respective path-link edges if and how they conflict with $B_i$. Once the required offset is computed, we shift $O(k)$ boxes by updating their lower left and upper right corners, as well as their origins. This is done in constant time per box and $O(k)$ time in total per iteration. So for all $k$ iterations the running time is $O(k^2)$. Finally, we obtain the embedding $\rho$ of $P'$ by computing the absolute coordinates of all vertices in $O(n)$ time. □

We can now conclude the running time for our simple path schematization algorithm by combining the results from Corollary 2 and Theorem 5.

**Corollary 3.** *The time complexity of the simple path schematization algorithm is in $O(k^2 + n^2)$.*

Since for practical purposes minimizing the total path length is sensible we additionally need to solve the linear programs described in Section 5.1.2. However, for the stated reasons in that section we cannot make any guarantees with respect to the quality of the solution.

Note that if the total number $I$ of conducted shift operations by the algorithm, is sufficiently small, we can do better than $O(k^2 + n^2)$ with the help of more sophisticated data structures. More specifically, if $I < k^2/\log k$ we can use fully dynamic data structures to achieve a time complexity of $O((I + k)\log k + n^2)$ for the part of the algorithm that is independent of the linear program.

As before we store for each bounding box its lower left and upper right coordinates. Additionally, we maintain in total four fully dynamic data structures; two for answering orthogonal line segment intersection queries, and the remaining two for answering orthogonal range reporting queries. One of each two stores the horizontal and the other the vertical segments of all already placed bounding boxes. During the step in which all conflicts are resolved, we use the those data structures to determine only those bounding boxes (and path-link edges) that really intersect a newly placed bounding box. Since this also changes the position of the horizontal and vertical segments of some the already bounding boxes we need to update the data structures accordingly.

We can use data structures proposed by Mortensen [27] to achieve this. Those data structures support insertions and deletions with worst-case time complexity in $O(\log n)$ and queries with worst-case time complexity in $O(\log n + J_i)$, where $J_i$ is the total number of reported segments. In total we need to shift a number $I_i \geq J_i$ of boxes, each of which can be done in $O(\log n)$ time by updating the appropriate data structures. Hence, each iteration requires $O(I_i \log n + \log n)$ time. The total time complexity is $O((I + k)\log k + n^2)$, and the total space requirement for the data structures is $O(n \log n/\log\log n)$.

## 6. A Mixed Integer Program for Path Schematization

From Corollary 1 we know that the Path Schematization Problem is NP-complete for every $d \geq 1$. So, unless P = NP, we cannot hope for an efficient exact algorithm to solve the problem. In Section 5 we showed that for a restricted class of paths, namely axis-monotone paths, the problem can be solved efficiently. However, the extension of this approach to general paths does not guarantee that the orthogonal order of the whole path is preserved. Thus, first we formulate the problem as a mixed integer linear program (MIP) and then justify our approach through an experimental study by showing that for the application of drawing sketches for real-world driving directions our MIP yields good results in reasonable time frames. The formulation of our MIP is similar to a MIP model for drawing octilinear metro maps [16].

### *6.1. Modeling*

Let $(P, \pi)$ be a given embedded graph, and let $d$ be a positive integer. Recall that for an embedding $\pi$ of $P$ and an edge $e$ of $P$ we denote by $\alpha_\pi(e)$ the direction of $e$ in $\pi$. Besides deciding whether $\mathcal{I}$ admits a valid $d$-schematization, we transform PSP into the following optimization problem in order to find a valid $d$-schematization that is as similar to the input path as possible.

**Problem 3** (General-Path Schematization Problem)**.** *Given a simple embedded path $(P = (V, E), \pi)$, an integer $d \geq 1$, and a minimum length $\ell_{\min}(e)$ for each $e \in E$, find a valid $d$-schematization $\rho$ such that*

*(i) for every edge $e \in E$ the deviation of $\alpha_\rho(e)$ from $\alpha_\pi(e)$ is minimal,*

*(ii) for every edge $e \in E$ the length $\ell(e)$ is at least a minimum length $\ell_{\min}(e) > 0$, and*

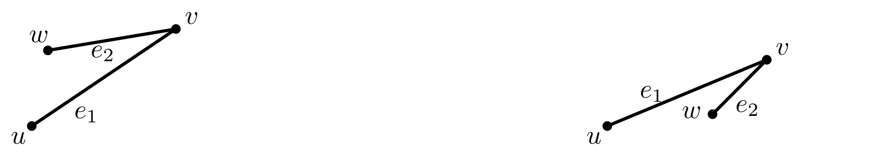
*(iii) the total length of the path $\sum_{e \in E} \ell(e)$ is minimized.*

In the following, we introduce our MIP that solves Problem 3 optimally.

*Coordinate System.* In order to handle the valid directions of each edge with respect to $\mathcal{C}_d$, we extend the standard Cartesian coordinate system to include one axis for every direction contained in $\mathcal{C}_d$ similarly to [16]. Therefore, we introduce continuous variables $\mathrm{pos}\colon V \times \mathcal{C}_d \to \mathbb{R}$ for every vertex and every possible direction. The value for each coordinate axis $\gamma \in \mathcal{C}_d$ is then defined by $\mathrm{pos}(v, \gamma) = \cos\gamma \cdot x(v) + \sin\gamma \cdot y(v)$. Note, that these are linear constraints, as the values of $\sin\gamma$ and $\cos\gamma$ are constants for fixed $d$. Moreover, our coordinate system allows us to introduce binary variables $\mathrm{dir}\colon E \times \mathcal{C}_d \to \{0, 1\}$ for deciding whether in the output embedding $\rho$ an edge $e \in E$ is schematized with direction $\gamma \in \mathcal{C}_d$, namely, if and only if $\mathrm{dir}(e, \gamma) = 1$. To enforce that every edge is schematized according to exactly one direction $\gamma \in \mathcal{C}_d$, we add $\sum_{\gamma \in \mathcal{C}_d} \mathrm{dir}(e, \gamma) = 1$ as a constraint for every edge $e \in E$.

*Orthogonal Order.* To maintain the orthogonal order between all pairs of vertices $v_i, v_j \in V$, we can make use of the following observation: Let $v_1 \prec \ldots \prec v_n$ be the input sorted with respect to the $x$-coordinates of the vertices (with ties broken arbitrarily). Then we introduce a linear number of orthogonal order constraints as follows. We set $\mathrm{pos}(v_i, 0°) \leq \mathrm{pos}(v_{i+1}, 0°)$ if $x_\pi(v_i) < x_\pi(v_{i+1})$ and $\mathrm{pos}(v_i, 0°) = \mathrm{pos}(v_{i+1}, 0°)$ if $x_\pi(v_i) = x_\pi(v_{i+1})$. Analogously, the same constraints are independently introduced for the $y$-coordinates of the vertices and the values $\mathrm{pos}(v, 90°)$ for all $v \in V$.

*Edge Directions and Lengths.* To ensure that every edge $e \in E$ is embedded according to the direction $\gamma$ for which $\mathrm{dir}(e, \gamma) = 1$ holds, we make use of our extended coordinate system in the following way. The edge $e = uv$ is embedded according to $\gamma \in \mathcal{C}_d$ if and only if $u$ and $v$ have the same coordinate on both orthogonal axes $\gamma + 90°$ and $\gamma + 270°$. Thus, for every $e = uv \in E$ and every $\gamma \in \mathcal{C}_d$ we add the constraints $\mathrm{pos}(v, \gamma + 90°) - \mathrm{pos}(u, \gamma + 90°) \leq M(1 - \mathrm{dir}(e, \gamma))$ and $\mathrm{pos}(v, \gamma + 270°) - \mathrm{pos}(u, \gamma + 270°) \leq M(1 - \mathrm{dir}(e, \gamma))$. Here, $M$ is a sufficiently large constant, which we choose to be the maximum possible distance in our coordinate space. This has the effect that the constraints are trivially satisfied for $\mathrm{dir}(e, \gamma) = 0$ and that they enforce the correct direction for $\mathrm{dir}(e, \gamma) = 1$. Additionally, we set the minimum length for every edge by adding constraints $\mathrm{pos}(v, \gamma) - \mathrm{pos}(u, \gamma) \geq -M(1 - \mathrm{dir}(e, \gamma)) + \ell_{\min}(e)$. Again, if $M$ is sufficiently large, these constraints are relevant only if $\mathrm{dir}(e, \gamma) = 1$. In order to minimize the total path length, we



(a) Input path. (b) Valid $d$-schematization for $d = 4$.

Figure 17: Example where edge order is violated

define variables len: $E \to \mathbb{R}_0^+$ as an upper bound on the length of each edge $e = uv \in E$ by introducing for every $\gamma \in \mathcal{C}_d$ a constraint $\text{pos}(v, \gamma) - \text{pos}(u, \gamma) \leq \text{len}(e)$. Note that minimizing $\text{len}(e)$ through the objective function will result in tight upper bounds on the actual edge lengths, i. e., $\text{len}(e) = \ell(e)$.

To minimize the deviation of the schematized direction of every edge $e \in E$ to its input direction, we add continuous variables dev: $E \to \mathbb{R}_0^+$ that represent the deviation cost of $e$ according to its embedding in $\rho$. We set $\text{dev}(e) = \sum_{\gamma \in \mathcal{C}_d} c(e, \gamma) \,\text{dir}(e, \gamma)$, where $c\colon E \times \mathcal{C}_d \to \mathbb{R}_0^+$ is some user-defined deviation cost function. In our implementation we set $c(e, \gamma)$ to the distance in terms of the number of directions in $\mathcal{C}_d$ between the selected direction and the direction in $\mathcal{C}_d$ that is closest to the input direction $\alpha_\pi(e)$.

*Edge Order.* Although the orthogonal order usually maintains the user's mental map appropriately, there can be cases where the orthogonal order alone is not sufficient. Consider the example displayed in Figure 17. The path in Figure 17(b) is a valid $d$-schematization of the input path for $d = 4$, but the order of the edges at vertex $v$ has changed. For route sketches this would mean that instead of a left turn, the schematized path depicts a right turn. This is clearly undesirable. To avoid this problem we iterate over all inner-path vertices and add constraints where necessary. For simplicity we disregard in the following the orientation of the path's edge and assume that each edge incident to a vertex is oriented away from the vertex. First, observe that an edge $e$ incident to a vertex $v$ must lie, in every valid $d$-schematization, in the same quadrant as in the input with respect to $v$. Hence, to avoid the discussed problem it is sufficient to consider only those vertices where its incident edges lie in the same quadrant. Now, consider an inner-path vertex $v$ with incident edges $e_1$ and $e_2$. Without loss of generality we assume that both edges lie in the first quadrant (to $v$'s upper right) and $e_1$ has, in the input, the larger direction. Then, for each of smallest the $d - 2$ diagonal directions $\gamma$ in the first quadrant we add the constraint $\text{dir}(e_2, \gamma) \leq \sum_{90° \geq \gamma' > \gamma} \text{dir}(e_1, \gamma')$. Finally, since $e_2$ must not be assigned the largest diagonal direction we also require that $\text{dir}(e_2, 90°/d \cdot (d - 1)) = 0$.

Further, to avoid overlapping of edges that are incident to the same vertex, we use a similar approach. Note that for edges $e_1$ and $e_2$ incident to the same vertex $v$ to overlap both edges are either in the same quadrant with respect to $v$ or in adjacent quadrants. For simplicity we assume again that for each vertex that we consider, both incident edges are oriented away from it. If both edges lie in the same quadrant with respect to $v$, then the constraints above already ensure that no overlap can occur. If the edges are in adjacent quadrants with respect to $v$, then we only require $\text{dir}(e_1, \gamma) + \text{dir}(e_2, \gamma) = 0$ for the direction $\gamma \in \mathcal{C}_d$

that is contained in both quadrants.

*Planarity.* In the output embedding $\rho$, edges may be stretched and embedded with directions different from the input embedding. Thus, we have to ensure that the output embedding is still crossing-free. Hence, we introduce two types of planarity constraints. First, for two adjacent edges $e_i = uv$ and $e_j = vw$ we observe that $e_i$ and $e_j$ only intersect, if $\gamma(e_i) = \gamma(e_j) + 180°$. Thus, for every pair of adjacent edges $e_i, e_j$ and every $\gamma \in \mathcal{C}_d$ we add a constraint $\mathrm{dir}(e_i, \gamma) + \mathrm{dir}(e_j, \gamma + 180°) \leq 1$.

For the remaining non-adjacent pairs of edges $e_i = u_i v_i, e_j = u_j v_j$ we make use of the following observation. The edges $e_i$ and $e_j$ do not intersect if there is at least one coordinate axis $\gamma \in \mathcal{C}_d$ according to which both $u_j$ and $v_j$ lie beyond $u_i$ and $v_i$. Thus, we introduce binary variables $\mathrm{sep}\colon E \times E \times \mathcal{C}_d \to \{0, 1\}$ selecting for each pair of non-adjacent edges the axis $\gamma$ for which the previous observation needs to be true. We can now enforce planarity by adding four constraints for every pair $e_i, e_j$ and every $\gamma \in \mathcal{C}_d$. We set $\mathrm{pos}(X, \gamma) - \mathrm{pos}(Y, \gamma) \geq -M(1 - \mathrm{sep}(e_i, e_j, \gamma)) + \delta$, where $X \in \{u_i, v_i\}$, $Y \in \{u_j, v_j\}$, and $\delta \in \mathbb{R}_0^+$ is a user-defined minimum distance to be kept between non-adjacent edge-pairs.

Note that this approach to enforce planarity requires $O(|E|^2)$ binary variables and constraints. However, in our experiments on real-world driving directions, we observe that most of the planarity constraints seem to be unnecessary in the sense that the optimal solution without planarity constraints already yields a crossing-free path. Thus, we may add planarity constraints in a lazy fashion by an iterative process: We start by computing a solution without planarity constraints. Then, we test whether the solution has any intersections and add planarity constraints for these specific conflicting edge pairs. We then recompute a solution with the new set of constraints until we finally obtain a crossing-free embedding $\rho$.

*Objective Function.* Subject to all the constraints described above we minimize the following objective function:

$$\text{Minimize} \qquad \sum_{e \in E} \mathrm{dev}(e) + \frac{1}{M} \sum_{e \in E} \mathrm{len}(e).$$

Note that the coefficient $1/M$ ensures that our primary goal is to minimize the total angular deviation as modeled by $\sum_{e \in E} \mathrm{dev}(e)$, while minimizing the total path length becomes the secondary goal.

## 7. Experiments.

In this section we give a brief experimental evaluation of both approaches we discussed in Sections 5 and 6. For the MIP approach we generated 1000 shortest paths (with respect to the travel time metric) in the German road network by selecting source and target vertices uniformly at random. Although our approach requires planar input graphs and a quickest path in a road network is not necessarily planar, we can simply planarize the input path by adding for each intersection of the path a vertex at the intersection point. However, our heuristic approach described in Section 5.2 can only handle simple input paths. Hence, we generate a second data set that contains quickest paths without intersections. For the second data set we again select source and target vertex in the German road network uniformly at random, discard a path if it is self-intersecting, and stop when we have 1000 quickest paths without self-intersections.

The average number of vertices of a path is between 995.2 and 998.3 vertices for both data sets, hence, a schematization would yield a route sketch with far too much detail. Therefore, in a preprocessing step, we simplify each path using the Douglas-Peucker algorithm [10] but keep all important vertices such as points of road category changes. Moreover, we shortcut self-intersecting subpaths whose lengths are below a threshold. We set this threshold to 200m in our experiments. For the SPS algorithm we used the linear program described in Section 5.1.2 to reduce the total path length of each axis-monotone path individually.

This is to remove over- or underpasses near slip roads of highways, which are considered irrelevant for route sketches or would be depicted as an icon rather than a loop. Figure 18 illustrates an example of a route sketch obtained by our MIP and as comparison a route sketch generated by the heuristic described in Section 5.2. It clearly illustrates how a schematic route sketch, unlike the geographic map, is able to depict both high-detail and low-detail parts of the route in a single picture.

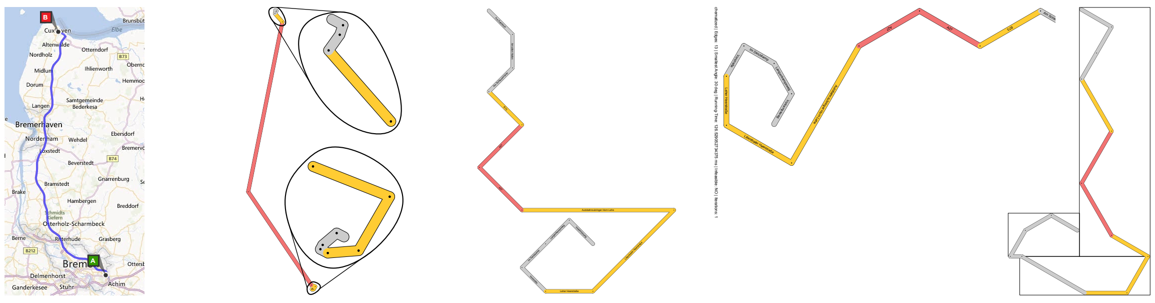
We performed our experiments on a single core of an AMD Opteron 2218 processor running Linux 2.6.34.10. The machine is clocked at 2.6 GHz, has 16 GiB of RAM and $2 \times 1$ MiB of L2 cache. Our implementation is written in C++ and was compiled with GCC 4.5.0 using optimization level 3. As MIP solver we use Gurobi 5.0.2.We performed our experiments on a single core of an AMD Opteron 2218 processor running Linux 2.6.34.10. The machine is clocked at 2.6 GHz, has 16 GiB of RAM and $2 \times 1$ MiB of L2 cache. Our implementation is written in C++ and was compiled with GCC 4.5.0 using optimization level 3. As MIP solver we use Gurobi 5.0.2. We have published the source code as a proof of concept implementation at `http://i11www.iti.kit.edu/_media/projects/routesketch.zip` under the GPL 3.0 License[1].

## *7.1. Case Study*

Comparing Figures 18(c) and 18(d) indicates that using the parameter $d = 2$ for the schematization process yields route sketches with a very high level of abstraction while using $d = 3$ results in route sketches which approximates the overall shape of the route much better. For example, the route sketch in Figure 18(c) seems to suggest that there is a 90° turn while driving on the highway but this is not the case. The route sketch depicted in Figure 18(c) resembles the original route much better. Generally, we found that using the parameter $d = 3$ yields route sketches with a high degree of readability. See Figure 18(d) for an example.

For comparison, we show the output computed by the heuristic method described in Section 5.2 in Figure 18(e). Recall that the heuristic subdivides a simple input path into maximal axis-monotone subpaths that are subsequently merged into a single route sketch. In order to avoid intersections of the bounding

[1]http://www.gnu.org/licenses/gpl-3.0



(a) (b) (c) (d) (e)

Figure 18: A sample route from Bremen to Cuxhaven in Germany: (a) output of Bing Maps, (b) simplified route after preprocessing, using $\varepsilon = 2^{-4}$ (with manually highlighted start and destionation), (c) output of our MIP for $d = 2$ (time to compute solution: 193 ms), (d) output of our MIP for $d = 3$ (time to compute solution: 126 ms), and (e) output of the heuristic from Section 5 (time to compute solution: 5.6 ms). Different colors depict different road categories. Note that in contrast to our solution, details at the beginning and the end of the route cannot be seen in the geographically accurate drawings.

boxes of the monotone subpaths, additional edges of appropriate length are inserted into the path; the orthogonal order is preserved only within the monotone subpaths. This may lead to undesired effects in sketches of non-monotone routes.

*Further Examples.* In Figures 19 and 20 are two example routes as displayed in Bing Maps (a), simplified route determined by the Douglas-Peucker algorithm (b), schematized by our MIP (c), and schematized by our SPS algorithm (d). Figure 19(d) shows an example of where the SPS algorithm resolved the intersection between the bounding boxes of both axis-monotone paths by adding a long horizontal path link edge. In Figure 20 we observe little difference between the optimal solution computed by our MIP and the solution obtained by the SPS algorithm. In this case no path link edge is necessary. Note that in this example the path computed by the SPS algorithm is shorter than the path computed by the MIP. This is due to the orthogonal order that here requires the grey horizontal line segment to have the same (or a larger) $y$-coordinate than the vertex connecting the two yellow line segments in the destination area of Konstanz.

## *7.2. Experimental Evaluation*

In this section we evaluate hard facts about two suggested approaches. We begin with the mixed integer linear program.

*Mixed Integer Linear Program.* In the first experiment we examine the performance of our MIP for $d = 3$ subject to the amount of detail of the route, as controlled by the distance threshold $\varepsilon$ between input and output path in the path simplification step. We also set the threshold for removing self-intersecting subpaths to $\varepsilon$. Tables 1(a) and 1(c) reports our experimental results for values of $\varepsilon$ between $2^{-1}$ and $2^{-5}$.

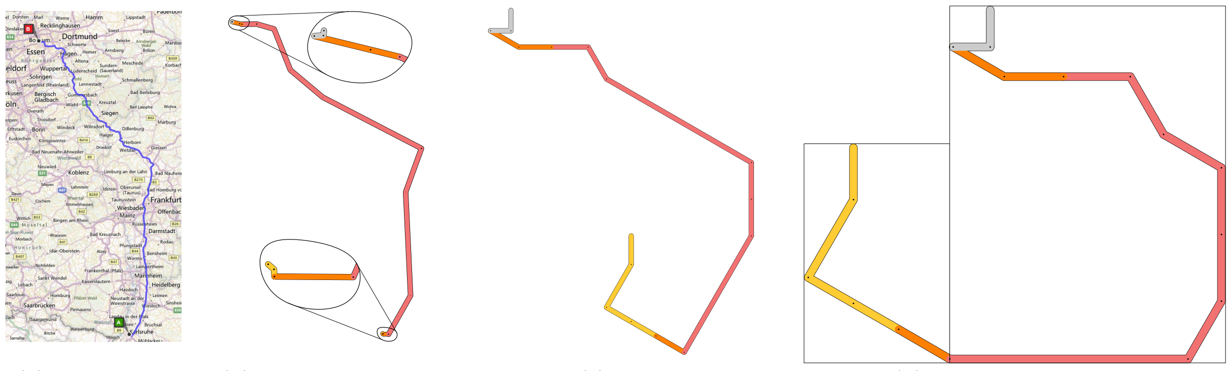

(a) Bing Maps (b) Simplified input (c) MIP (d) SPS algorithm

Figure 19: Route from Bochum to Karlsruhe. Parameters used: $\varepsilon = 2^{-4}$, and $d = 3$. Time to compute solutions with MIP and SPS algorithm are 262ms and 5.8ms, respectively.

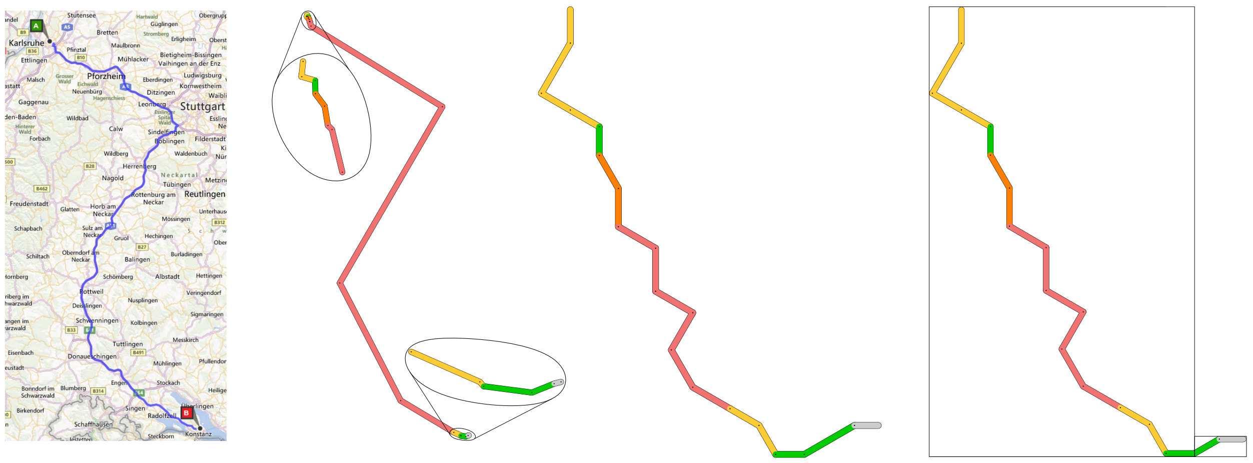

(a) Bing Maps (b) Simplified input (c) MIP (d) SPS algorithm

Figure 20: Route from Karlsruhe to Konstanz. Parameters used: $\varepsilon = 2^{-3}$ and $d = 3$. Time to compute MIP solution: 262ms. Time to compute with heuristic: 5.23ms.

Table 1: Results obtained by running 1000 random queries in the German road network and schematizing the simplified paths with our MIP. The tables reports average path lengths (number of vertices), percentage of infeasible instances, average number of iterations, and average running times. The Tables (a) and (b) shows the result for the data sets where self-intersecting input paths are planarized, while Tables (c) and (d) shows the result for the data sets where input paths are non self-intersecting.

| $\varepsilon$ | ~length | % inf | ~iter | ~time [ms] |
|---|---|---|---|---|
| $2^{-1}$ | 19.4 | 0.40 | 1.06 | 141.77 |
| $2^{-2}$ | 19.8 | 0.40 | 1.06 | 156.14 |
| $2^{-3}$ | 20.8 | 0.40 | 1.07 | 165.76 |
| $2^{-4}$ | 23.4 | 0.40 | 1.06 | 206.60 |
| $2^{-5}$ | 30.6 | 0.40 | 1.06 | 373.96 |

(a) Varying $\varepsilon$ with $d = 3$ (with intersections).

| $d$ | % inf | ~iter | ~time [ms] |
|---|---|---|---|
| 1 | 56.20 | 1.21 | 32.62 |
| 2 | 1.50 | 1.11 | 125.56 |
| 3 | 0.40 | 1.07 | 165.76 |
| 4 | 0.40 | 1.05 | 291.47 |
| 5 | 0.40 | 1.05 | 723.84 |

(b) Varying $d$ with $\varepsilon = 2^{-3}$ (with intersections). The average path length after simplifying with is 20.8 vertices.

| $\varepsilon$ | ~length | % inf | ~iter | ~time [ms] |
|---|---|---|---|---|
| $2^{-1}$ | 19.3 | 0.00 | 1.05 | 132.29 |
| $2^{-2}$ | 19.7 | 0.00 | 1.05 | 142.73 |
| $2^{-3}$ | 20.7 | 0.00 | 1.05 | 155.69 |
| $2^{-4}$ | 23.2 | 0.00 | 1.05 | 186.04 |
| $2^{-5}$ | 30.4 | 0.00 | 1.05 | 358.77 |

(c) Varying $\varepsilon$ with $d = 3$ (no intersections).

| $d$ | % inf | ~iter | ~time [ms] |
|---|---|---|---|
| 1 | 52.40 | 1.24 | 35.49 |
| 2 | 1.00 | 1.09 | 121.04 |
| 3 | 0.00 | 1.05 | 155.69 |
| 4 | 0.00 | 1.04 | 176.15 |
| 5 | 0.00 | 1.04 | 188.05 |

(d) Varying $d$ with $\varepsilon = 2^{-3}$ (no intersections). The average path length after simplifying with is 20.7 vertices.

For the first data set with decreasing $\varepsilon$, the lengths of the paths increase from 19.4 vertices to 30.6 vertices on average. This correlates with the running time of our MIP which is between 141.77 ms and 373.96 ms for the instances. We observe that a value of $\varepsilon = 2^{-3}$ is a good compromise between computation time and amount of detail in practice. Further, adding planarity constraints in a lazy fashion pays off since we need less than 1.14 iterations on average. Thus, most of the paths admit a planar schematization without any planarity constraints at all. The number of infeasible instances, i. e., paths without a valid 3-schematization, is 0.4 %. Finally, there is very little difference between the results for the first data set reported in Table 1(a) and for the second data set reported in Table 1(c).

The second experiment evaluates the performance of our MIP when using different values of $d$, see Table 1(b) and Table 1(d). We fix $\varepsilon = 2^{-3}$. While for rectilinear drawings with $d = 1$ we need 32.62 ms to compute a solution, the running time increases to 723.84 ms when using $d = 5$. Interestingly, more than half of the paths do not have a valid rectilinear schematization. By allowing one additional diagonal direction

($d = 2$), the number of infeasible instances significantly decreases to roughly 1.5 %. Adding further diagonal directions, i. e., increasing $d$ to a value larger than 2, does not seem to have a significant impact on the percentage of infeasible instances. We observe that in terms of infeasible instances and average number of iterations there is little difference between the two tested data sets. However, the running time for $d = 5$ for the first data set is 723.84 ms while it is only 188.05 ms for the second data set. This is most likely caused by more difficult instances that are not included in the second data sets because of self-intersections.

It seems surprising that for $d = 1$ there roughly half of all paths in the evaluation do not have a valid rectilinear schematization. This is due to the very limited schematization options coupled with the constraints posed by maintaining the orthogonal order. There are many examples of even very short paths which do not have a valid rectilinear schematization. One such example is depicted in Figure 21. If the edge $e_1$ is assigned direction 90°, then all vertices must share the same $x$-coordinate, but $w$ must be below $v$ which forces $e_1$ and $e_2$ to overlap. Otherwise, if $e_1$ is assigned 0°, then all vertices must share the same $y$-coordinate. By similar argumentation as before, the edges $e_1$ and $e_2$ must overlap yielding a non-valid rectilinear schematization. Adding only one diagonal direction to the set $\mathcal{C}$ resolves this problem. Hence, it seems that the problem of finding a valid schematization becomes significantly easier when diagonal directions are permitted.

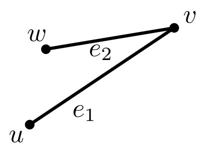


Figure 21: Path that has no valid rectilinear schematization.

*SPS algorithm.* Similar to the first experiment, in the third experiment we examine the performance of our heuristic approach described in Section 5.2, the simple path schematization (SPS) algorithm. Again, we set $d = 3$ and vary the value of $\varepsilon$. Note that contrary to the MIP, which always computes an optimal solution (if there exists one), for our heuristical approach there are no infeasible instances (given that the input paths are simple). However, there are two possible problems by the output generated with the algorithm. First, there is no guarantee that the orthogonal order of the input path is preserved. Second, during the process we may need to add several edges to the path. Hence, we report additionally to average running time of the algorithm two parameters: average percentage of vertex-pairs for which the orthogonal order is violated and the average number of edges that is added to the path. Table 2 reports our experimental results for values of $\varepsilon$ between $2^{-1}$ and $2^{-5}$.

We observe that with decreasing values of $\varepsilon$ there is a slight increase in the average number of monotone paths per quickest paths from 3.35 to 4.01. This correlates with a slight increase in the average number of added path link edges by our algorithm, which increases from 0.55 to 0.60. Note that this number may seem surprisingly low since our algorithm adds at least two edges to connect two axis-monotone paths. However, in our experimental analysis we observed that often the bounding boxes of the axis-monotone paths do not overlap, which means the path link edges have length 0 and thus we can omit them completely. On the

Table 2: Results obtained by running 1000 simple paths obtained by random quickest path queries in the German road network and schematizing the simplified paths with our SPS algorithm. The table reports the average number of axis-monotone paths per quickest path, the average number of added path link edges to the path, the average part of the total path length that is contributed by the path link edges, average percentage of vertex pairs for which the orthogonal order is maintained, and average running time. We consider varying values for $\varepsilon$ with $d = 3$.

| $\varepsilon$ | ∼mon. path. | ∼add. edges | ∼add. edge length[%] | ∼orth. order[%] | ∼time [ms] |
|---|---|---|---|---|---|
| $2^{-1}$ | 3.35 | 0.55 | 8.1 | 92.51 | 5.16 |
| $2^{-2}$ | 3.40 | 0.56 | 8.1 | 92.48 | 5.22 |
| $2^{-3}$ | 3.46 | 0.57 | 7.6 | 93.12 | 5.89 |
| $2^{-4}$ | 3.57 | 0.57 | 7.5 | 93.94 | 5.87 |
| $2^{-5}$ | 4.01 | 0.60 | 6.0 | 95.69 | 6.57 |

other hand, we observe that if path link edges are added that they contribute a non-negligible portion to the total path length (between 6% and 8.1%). Further, we note that the number of vertex pairs for which the orthogonal order is preserved also increases slightly with decreasing values of $\varepsilon$ from 92.51% to 95.69%. This is most likely due to the increased number of vertices in the input path which decreases the effect of a single vertex that violates the orthogonal order of the input. Finally, we observe that the running time also increases slightly with decreasing values of $\varepsilon$ from 5.16 ms to 6.57 ms which is caused by the increased number of vertices in the input.

We have seen that with our MIP formulation, which guarantees an optimal result, we can obtain good schematizations in a reasonable time frame. Further, the number of infeasible instances (i.e., instances for which there exists no solution) is relatively small. If a fast execution time is more important than optimality, the SPS algorithm presents a viable alternative that produces good results, and in our experiments had an average running time between 5 ms and 7 ms, while the MIP required for the same routes between 132.29 ms and 358.77 ms. However, the SPS algorithm can only be used for simple input paths. Another downside of the SPS algorithm is the proportion of the total path length that is contributed by the path link edges. To avoid this it may be a good alternative to use an approach to resolve conflicts that is not solely based on the bounding boxes of the axis-monotone paths.

## 8. Conclusion

Motivated by drawing route sketches in road networks, we studied the problem of $d$-regular schematization of embedded paths (PSP). In our problem definition we aimed for two main goals: In order to preserve the user's mental map we used the concept of orthogonal order, and to reduce the visual complexity we restricted the valid edge directions to integer multiples of $(90/d)^\circ$. We analyzed the complexity of the problem and showed that PSP is NP-complete for $d \geq 2$. In combination with a previous result of Brandes and Pampel [9, 18] it follows that PSP is NP-complete for every integer $d \geq 1$. Unfortunately, our proof does not work anymore when the input points are in general position. This is in contrast to the proof by Brandes and Pampel. Hence, the complexity of the Path Schematization Problem for $d > 1$ when all input points are in general position is open, but in general the problem is NP-complete.

Nonetheless, we presented two approaches to generate route sketches. The first one is a polynomial-time algorithm that computes for axis-monotone paths and arbitrary integer $d$ a valid $d$-schematization with minimum schematization cost, for any $d$. We used this algorithm to derive a heuristic to generate $d$-schematizations for arbitrary simple paths. An experimental evaluation showed that this approach can very quickly generate outputs with only limited violations of the orthogonal order. However, it is restricted to simple input paths. Our second practical approach to generate route sketches uses mixed integer linear programming (MIP). To generate readable drawings, we minimized the total path length in the objective function of our MIP while ensuring a certain minimum length for each individual edge. Finally, we conducted an experimental evaluation on real-world data in the German road network for both approaches. They revealed that we are indeed able to compute visually appealing solutions of the PSP within approximately 1 sec for reasonable values of $d \leq 5$.

Although maintaining the orthogonal order of the vertices is a common way to preserve the mental map of the user, the orthogonal order has some downsides. It makes it difficult to find valid schematizations (it is NP-complete to decide whether or not a path has a schematization that maintains the orthogonal order) and sometimes the orthogonal order prohibits schematizations which appear to be sensible. To overcome this problem it might make sense to investigate relaxed variants of the orthogonal order, in which a vertex only affects vertices that are "close" to it. Thus, the orthogonal order is locally maintained, which might be sufficient to preserve a global mental map. Since we use the orthogonal order only for maintaining the mental map, an interesting direction is to study alternative ways to model mental map preservation.

## References

[1] D. Delling, A. Gemsa, M. Nöllenburg, T. Pajor, Path schematization for route sketches, in: Proc. 12th Scandinavian Symposium & Workshops on Algorithm Theory (SWAT'10), Vol. 6139 of Lecture Notes in Computer Science, Springer, 2010, pp. 285–296. doi:10.1007/978-3-642-13731-0_27.

[2] A. Gemsa, M. Nöllenburg, T. Pajor, I. Rutter, On d-regular schematization of embedded paths, in: Proc. 37th Int. Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'11), Vol. 6543 of Lecture Notes in Computer Science, Springer, 2011, pp. 260–271. doi:10.1007/978-3-642-18381-2_22.

[3] S. Cabello, M. d. Berg, S. v. Dijk, M. v. Kreveld, T. Strijk, Schematization of road networks, in: Proc. 17thAnn. ACM Symposium on Computational Geometry (SoCG'01), ACM Press, 2001, pp. 33–39. doi:10.1145/378583.378609.

[4] U. Brandes, M. Eiglsperger, M. Kaufmann, D. Wagner, Sketch-driven orthogonal graph drawing, in: S. Kobourov, M. Goodrich (Eds.), Proc. 10th Int. Symposium on Graph Drawing (GD'02), Vol. 2528 of Lecture Notes in Computer Science, Springer, 2002, pp. 1–11. doi:10.1007/3-540-36151-0_1.

[5] J. Branke, Dynamic graph drawing, in: M. Kaufmann, D. Wagner (Eds.), Drawing Graphs: Methods and Models, Vol. 2025 of Lecture Notes in Computer Science, Springer, 2001, Ch. 9, pp. 228–246. doi:10.1007/3-540-44969-8_9.

[6] K. Misue, P. Eades, W. Lai, K. Sugiyama, Layout adjustment and the mental map, Journal of Visual Languages and Computing 6 (2) (1995) 183–210. doi:10.1006/jvlc.1995.1010.

[7] K. Hayashi, M. Inoue, T. Masuzawa, H. Fujiwara, A layout adjustment problem for disjoint rectangles preserving orthogonal order, in: S. H. Whitesides (Ed.), Proc. 6th Int. Symposium on Graph Drawing (GD'98), Vol. 1547 of Lecture Notes in Computer Science, Springer, 1998, pp. 183–197. doi:10.1002/scj.1104.

[8] T. Dwyer, Y. Koren, K. Marriott, Stress majorization with orthogonal ordering constraints, in: P. Healy, N. S. Nikolov (Eds.), Proc. 13th Int. Symposium on Graph Drawing (GD'05), Vol. 3843 of Lecture Notes in Computer Science, Springer, 2006, pp. 141–152. doi:10.1007/11618058_14.

[9] U. Brandes, B. Pampel, On the hardness of orthogonal-order preserving graph drawing, in: I. Tollis, M. Patrignani (Eds.), Proc. 16th Int. Symposium on Graph Drawing (GD'08), Vol. 5417 of Lecture Notes in Computer Science, Springer, 2009, pp. 266–277. doi:10.1007/978-3-642-00219-9_25.

[10] D. H. Douglas, T. K. Peucker, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, Cartographica 10 (2) (1973) 112–122. doi:10.3138/FM57-6770-U75U-7727.

[11] D. Merrick, J. Gudmundsson, Path simplification for metro map layout, in: M. Kaufmann, D. Wagner (Eds.), Proc. 14th Int. Symposium on Graph Drawing (GD'06), Vol. 4372 of Lecture Notes in Computer Science, Springer, 2007, pp. 258–269. doi:10.1007/978-3-540-70904-6_26.

[12] G. Neyer, Line simplification with restricted orientations, in: F. K. Dehne, A. Gupta, J.-R. Sack, R. Tamassia (Eds.), Proc. 6th Int. Workshop on Algorithms and Data Structures (WADS'99), Vol. 1663 of Lecture Notes in Computer Science, Springer, 1999, pp. 13–24. doi:10.1007/3-540-48447-7_2.

[13] M. Agrawala, C. Stolte, Rendering effective route maps: Improving usability through generalization, in: E. Fiume (Ed.), Proc. 28th Ann. Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'01), ACM, 2001, pp. 241–249. doi:10.1145/383259.383286.

[14] J. Kopf, M. Agrawala, D. Bargeron, D. Salesin, M. Cohen, Automatic generation of destination maps, ACM Trans. Graph. 29 (6) (2010) 158:1–158:12. doi:10.1145/1882261.1866184.

[15] M. Nöllenburg, Automated drawing of metro maps, Tech. Rep. 2005-25, Fakultät für Informatik, Universität Karlsruhe (2005).
URL http://digbib.ubka.uni-karlsruhe.de/volltexte/1000004123

[16] M. Nöllenburg, A. Wolff, Drawing and Labeling High-Quality Metro Maps by Mixed-Integer Programming, IEEE Transactions on Visualization and Computer Graphics 17 (5) (2011) 626–641. doi:10.1109/TVCG.2010.81.

[17] A. Wolff, Drawing subway maps: A survey, Informatik - Forschung und Entwicklung 22 (1) (2007) 23–44. doi:10.1007/s00450-007-0036-y.

[18] U. Brandes, B. Pampel, Orthogonal-ordering constraints are tough, Journal of Graph Algorithms and Applications 17 (1) (2013) 1–10. doi:10.7155/jgaa.00281.

[19] B. Speckmann, K. Verbeek, Homotopic rectilinear routing with few links and thick edges, in: A. López-Ortiz (Ed.), LATIN 2010: Theoretical Informatics, Vol. 6034 of Lecture Notes in Computer Science, Springer, 2010, pp. 468–479. doi:10.1007/978-3-642-12200-2_41.

[20] K. Verbeek, Homotopic $\mathcal{C}$-oriented routing, in: Proc. 20th Int. Symposium on Graph Drawing (GD'12), Lecture Notes in Computer Science, Springer, 2013, pp. 272–278. doi:10.1007/978-3-642-36763-2_24.

[21] K. Buchin, W. Meulemans, B. Speckmann, A new method for subdivision simplification with applications to urban-area generalization, in: Proc. 19th ACM SIGSPATIAL Int. Conference on Advances in Geographic Information Systems (GIS'11), ACM, 2011, pp. 261–270. doi:10.1145/2093973.2094009.

[22] M. Berg, A. Khosravi, Optimal binary space partitions in the plane, in: M. Thai, S. Sahni (Eds.), Proc. 6th Ann. Int. Conference on Computing and Combinatorics (COCOON 2010), Vol. 6196 of Lecture Notes in Computer Science, Springer, 2010, pp. 216–225. doi:10.1007/978-3-642-14031-0_25.

[23] P. Vaidya, Speeding-up linear programming using fast matrix multiplication, in: Proc. 30th Ann. Symposium on Foundations of Computer Science (FOCS'89), 1989, pp. 332–337. doi:10.1109/SFCS.1989.63499.

[24] P. A. Beling, Exact algorithms for linear programming over algebraic extensions, Algorithmica 31 (4) (2001) 459–478. doi:10.1007/s00453-001-0049-z.

[25] D. H. Lehmer, A note on trigonometric algebraic numbers, The American Mathematical Monthly 40 (3) (1933) 165–166. URL http://www.jstor.org/stable/2301023

[26] E. Swift, Discussions: Note on trigonometric functions, The American Mathematical Monthly 29 (10) (1922) 404–405. URL http://www.jstor.org/stable/2299031

[27] C. W. Mortensen, Fully-dynamic two dimensional orthogonal range and line segment intersection reporting in logarithmic time, in: Proc. 14th Ann. ACM-SIAM Symposium on Discrete Algorithms, SODA '03, Society for Industrial and Applied Mathematics, 2003, pp. 618–627.
URL http://dl.acm.org/citation.cfm?id=644108.644210