A Hierarchy-Focused Algorithm for Drawing Single Line Diagrams of Power Grids

Gabriel Ott
Carl von Ossietzky University Oldenburg
Oldenburg, Germany
Email: gabriel.ott@uol.de

Eric MSP Veith
OFFIS – Institute for Information Technology
R&D Division Energy
Oldenburg, Germany
Email: eric.veith@offis.de

***Abstract*—Visualizing power grids has always been important: Not only to show their current state, but especially to help operators grasp problems and find solutions quickly. To this end, the single-line diagram is still the most dominant method. However, many current diagrams are still drawn manually and only updated with values at run-time. While many approaches for automatic drawing exist, none of them respects voltage levels as important hierarchical organizational aspect. Thus, cross-level state visualization is still not automatic. To this end, we present a two-stage algorithm for automatic drawing of single-line diagrams.**

***Keywords*—power grid topology; grid topology visualization; control room software**

I. Introduction

Algorithms for creating visualizations of power grids are a well-studied field. Fischer et al. [1] provide an overview over the numerous forms of visualizations that exist in general. Notably, there is not one type that would be fitting for all kinds of tasks; rather, the form of displaying information to the user must be based upon the user's and the situation's specific needs. For example, choosing to include geospatial information or color-coding fundamentally changes requirements to the algorithm.

While Fischer et al. [1] note that modern grid operation software suites fail to leverage many of the more advanced visualization techniques developed in academia, one particular kind of diagram persists: The *single line diagram*. It is ubiquitous in modern control rooms. Since single-line diagrams are essentially orthogonal graph drawing problems, algorithms can draw from publications made as old as 1987 [2].

However, while classical orthogonal graph drawing algorithms optimize for aesthetic criteria, such as bend minimization, crossing reduction, and area compactness, they neglect a fundamental aspect critical to power system operation: voltage-level hierarchy. In power grids, the hierarchical organization of voltage levels—from high-voltage transmission networks (e. g., $400\,\mathrm{kV}$, $220\,\mathrm{kV}$) down through distribution levels ($110\,\mathrm{kV}$, $20\,\mathrm{kV}$) to low-voltage feeders—directly reflects energy flow direction and failure propagation patterns. Grid operators rely on this hierarchy to quickly assess which components depend on which feeders during fault analysis and to understand cascading failure risks.

Existing automatic layout algorithms for single-line diagrams fall into three main categories, none of which adequately preserve voltage hierarchy. *Force-directed methods* optimize for visual balance and minimal edge crossings but produce non-orthogonal layouts with arbitrary component placement that obscures hierarchical relationships. *Geographic approaches* prioritize spatial fidelity, maintaining real-world locations of substations at the expense of clear hierarchical structure. *Generic hierarchical algorithms* assign layers based on graph topology (e. g., longest paths, topological ordering) rather than domain semantics, potentially placing $400\,\mathrm{kV}$ and $20\,\mathrm{kV}$ components on the same visual layer simply because they are topologically adjacent.

This gap—i. e., algorithms not preserving the voltage hierarchy—has practical consequences in control room operations. When operators face time-critical situations, such as equipment failures, overload conditions, or cascading outages, they need immediate visual clarity about which voltage levels are affected and how failures might propagate through the hierarchy. Layouts that prioritize compactness or geographic accuracy over hierarchical structure force operators to mentally reconstruct the voltage-level dependencies, increasing cognitive load and response time [3]. Such a visualization is particularly important considering recent advances in power grid automation, where algorithms based on Artificial Intelligence (AI) (e. g., learning agents) are able to support operators in handling emergency situations [4], since these agents are most probably operating on a hierarchical representation of the grid, too.

In this paper, we present a hierarchy-focused algorithm specifically designed to preserve voltage-level structure in single-line diagrams of power grids. It consists of two distinct phases, node placement via the Semi-Automatic Ordering Algorithm (SAOA) algorithm and line drawing via the *Rollwire* algorithm. Unlike existing approaches, our approach treats voltage hierarchy as the primary organizing principle rather than as a secondary aesthetic concern. The algorithm performs a hierarchical decomposition based on voltage levels, assigns components to vertical layers corresponding to their voltage ratings, and applies orthogonal layout techniques within each layer while maintaining clear visual separation between levels. This approach ensures that operators can immediately identify high-voltage transmission infrastructure, mid-level distribution networks, and low-voltage feeders without searching through the diagram.

The remainder of this paper is organized as follows: We discuss related works in Section II. We describe an ordering algorithm that establishes a parent-child element ordering for

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

drawing purposes, called *Semi-Automatic Ordering Algorithm* (SAOA), in Section III. We detail how the layouting itself is done in Section IV. Before concluding, we provide a discussion of the algorithm in Section V.

II. Related Work

Classical orthogonal graph drawing focuses primarily on aesthetic criteria such as bend minimization, area reduction, and crossing minimization. The seminal work of Tamassia [2] established the network flow approach for computing bend-minimal orthogonal representations of plane 4-graphs, achieving polynomial-time optimality for bend count. Biedl et al. [5] improved upon this with an $\mathcal{O}(n)$ algorithm producing at most $2n + 2$ bends in $n \times n$ area, while Freivalds et al. [6] developed a grid-based algorithm that minimizes total edge length through iterative local optimization and quadratic programming compaction. More recently, Kieffer et al. [7] introduced Human-like Orthogonal Network Layout (HOLA), a user study-driven algorithm that identified nine aesthetic criteria valued by humans, including tree-outside placement, compactness, symmetry, and "gridiness." HOLA's four-stage approach—decomposing graphs into cores and trees, stress-minimization layout, tree placement, and fine-tuning—produces more human-like results than traditional topology-shape-metrics approaches. However, none of these algorithms consider domain-specific hierarchical constraints. They optimize for visual aesthetics without preserving the semantic structure critical for power grid interpretation, where voltage hierarchy directly impacts operational understanding.

The Sugiyama framework [8] and its descendants provide principled approaches to layered graph layout through three phases: layer assignment, crossing reduction, and coordinate assignment. Brandes et al. [9] contributed fast horizontal coordinate assignment with vertical alignment in linear time, becoming widely adopted event in tools like *yFiles*. However, these methods assign hierarchy based on graph topology (e. g., longest path, topological ordering) rather than domain semantics. In power systems, the meaningful hierarchy is determined by voltage levels and energy flow direction, not graph structure. A substation at $400\,\mathrm{kV}$ must be visually separated from $110\,\mathrm{kV}$ equipment regardless of their topological distance. Applying generic hierarchical layouts to power grids fails to emphasize this critical semantic dimension, potentially placing high-voltage and low-voltage components on the same visual layer simply because they are topologically adjacent.

Automated generation of power system visualizations has received limited attention despite its operational importance. Birchfield et al. [10] developed force-directed and greedy approaches for geographic one-line diagrams, using Delaunay triangulation for line routing to maintain geographic context while minimizing substation overlap. Their method prioritizes spatial fidelity over hierarchical clarity. Earlier work by Raman et al. [11] provided foundational automatic generation algorithms, while Lendák et al. [12] applied force-directed methods requiring 300 iterations without orthogonal constraints, resulting in non-rectangular edges unsuitable for standard engineering practice. More recent efforts have addressed specific subtasks: Hong et al. [13] proposed orthogonal layouts optimized for connectivity visualization but not hierarchy preservation; Yang et al. [14] introduced graph partitioning for scalable layouts of large distribution networks; and Sen et al. [15] developed incremental update methods for radial distribution systems with tree structures. Nagendra Rao et al. [16] specifically addressed distribution system diagram generation using tree drawing algorithms, noting that radial networks naturally form tree topologies rooted at substation transformers. They compared spring embedder, controlled spring embedder, and visibility-based approaches, concluding that existing general-purpose algorithms produce unsatisfactory layouts for power systems. However, their approach still applies graph-theoretic thinking by using the distance from the root element (i. e., the feeder) as metric, which in turn means that their algorithm is not suitable for visualizations that display several connected voltage levels. Critically, all these methods treat power networks as generic graphs, optimizing for compactness, crossing reduction, or geographic accuracy while neglecting the voltage hierarchy that operators rely on to assess failure propagation and system dependencies.

The literature reveals a fundamental gap: no existing algorithm explicitly preserves voltage-level hierarchy as a primary layout constraint. General orthogonal algorithms prioritize aesthetics; hierarchical methods impose topology-based layers; and power system tools focus on geographic or connectivity representation. Our algorithm addresses this gap by making parent-child voltage dependencies the central organizing principle, ensuring that operators can immediately identify which components depend on which feeders—critical information during fault analysis that is obscured by conventional layout algorithms optimized for different objectives. This hierarchy-first approach, combined with the semi-automatic philosophy allowing manual refinement, distinguishes our algorithm from all prior work in the intersection of graph drawing and power system visualization.

III. Semi-Automatic Ordering Algorithm

A. Overview and Definitions

Our algorithm works in a two-stage approach. First, the *SAOA* algorithm is responsible for placing nodes. Second, the actual drawing of nodes and especially the connecting lines is then done in a second step by the *Rollwire* layouting algorithm (cf. Section IV).

SAOA operates in four phases: (1) initialization, (2) parent-child assignment, (3) iterative space propagation, and (4) absolute position computation. The algorithm ensures that parent nodes are always positioned above their children, preserving the voltage-level hierarchy visually. Figure 1 shows the effect of following SAOA as it is described in the following sections.

Formally, let $G = (V, E)$ be a directed graph representing the power grid topology, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of grid components (buses, transformers, generators, loads) and $E \subseteq V \times V$ is the set of directed edges representing electrical connections. For transformers, edge direction is determined

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

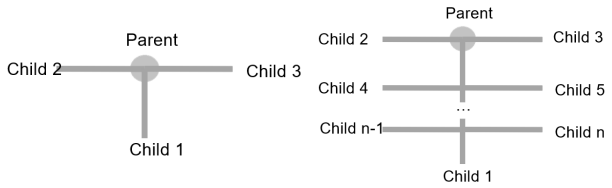

Figure 1. Placement of nodes as following our Semi-Automatic Ordering Algorithm

by voltage level: edges point from the high-voltage (upstream) side to the low-voltage (downstream) side, establishing the voltage hierarchy.

We define:

- **Parent function** $\pi : V \to V \cup \{\perp\}$: For each node $v \in V$, $\pi(v)$ denotes its unique parent node. If v is a root node, $\pi(v) = \perp$.
- **Children set** $C(v) = \{u \in V : \pi(u) = v\}$: The set of all children of node v.
- **Position** $\boldsymbol{p}(v) = (x_v, y_v) \in \mathbb{Z}^2$: The absolute grid coordinates of node v.
- **Relative offset** $\boldsymbol{\delta}(v) = (\delta_x(v), \delta_y(v)) \in \mathbb{Z}^2$: The position of v relative to its parent, such that:

$$\boldsymbol{p}(v) = \boldsymbol{p}(\pi(v)) + \boldsymbol{\delta}(v) \ . \tag{1}$$

- **Space requirements** $\boldsymbol{s}(v) = (s_L(v), s_R(v), s_B(v)) \in \mathbb{N}_0^3$: The space required by node v and all its descendants in the left, right, and below directions, respectively.
- **Hierarchical depth** $d(v)$: The length of the path from the root to v, with $d(v_{\text{root}}) = 0$.

B. Phase 1: Initialization

Input: Graph $G = (V, E)$ with voltage-level annotations for transformer edges.

Output: Initial parent-child relationships and default offsets.

1) For each node $v \in V$, initialize $\boldsymbol{\delta}(v) \leftarrow (0, 0)$ and $\boldsymbol{s}(v) \leftarrow (0, 0, 0)$.
2) For each edge $(u, v) \in E$, set $\pi(v) \leftarrow u$ and add v to $C(u)$.
3) For nodes with multiple potential parents, assign to the first parent with available capacity; flag conflicts for manual review.

C. Phase 2: Child Placement

For each node v with children $C(v) = \{c_1, c_2, \ldots, c_k\}$, assign initial offsets based on child index. Let $b(v) \in \{0, 1\}$ indicate whether the parent-of-parent occupies a lateral position (i. e., $\delta_x(\pi(v)) \neq 0$):

$$\boldsymbol{\delta}(c_i) = \begin{cases} (0, 1) & \text{if } i = 1 \text{ (first child: below)} \\ (-1, 0) & \text{if } i = 2 \text{ (second child: left)} \\ (+1, 0) & \text{if } i = 3 \text{ (third child: right)} \end{cases} \tag{2}$$

For $k > 3$ children, the first child is shifted downward to level $\ell = \lceil \frac{k+b(v)}{2} \rceil$, and additional children are placed alternately left and right at decreasing levels:

Input : Nodes V with initial offsets $\boldsymbol{\delta}$ and parent assignments π

Output : Final offsets $\boldsymbol{\delta}$ ensuring overlap-free placement

```
repeat
    changed ← false
    foreach v ∈ V do
        // Aggregate children's space requirements:
        foreach direction d ∈ {L, R, B} do
            s_d(v) ← max_{c∈C(v)} (s_d(c) + |δ_d(c)|)
        end
        // Update offset to parent based on sibling space:
        if π(v) ≠ ⊥ then
            foreach direction d ∈ {L, R, B} do
                σ ← s(siblings in opposite direction)  δ'_d ← s_d(v) + σ
                if δ'_d ≠ δ_d(v) then
                    δ_d(v) ← δ'_d
                    changed ← true
                end
            end
        end
    end
until ¬changed;
```

Figure 2. SAOA Iterative Space Propagation

$$\boldsymbol{\delta}(c_1) = (0, \ell), \tag{3}$$

$$\boldsymbol{\delta}(c_j) = \begin{cases} (-1, \ell - \lfloor j/2 \rfloor) & \text{if } j \text{ even} \\ (+1, \ell - \lfloor j/2 \rfloor) & \text{if } j \text{ odd} \end{cases} \quad \text{for } j > 1 \tag{4}$$

If $\pi(v)$ occupies the left or right position relative to v, that position is reserved and children are assigned to the next available slot.

D. Phase 3: Iterative Space Propagation

The core of SAOA is an iterative propagation of space requirements from leaves to root. Each node computes its space needs based on its children's requirements and adjusts its offset to its parent accordingly.

E. Phase 4: Absolute Position Computation

Once offsets have converged, absolute positions are computed by traversing from root to leaves:

$$\mathbf{p}(v) = \begin{cases} \boldsymbol{p}_0 & \text{if } \pi(v) = \perp \ , \\ \mathbf{p}(\pi(v)) + \boldsymbol{\delta}(v) & \text{otherwise,} \end{cases} \tag{5}$$

where $\boldsymbol{p}_0$ is a predefined root position.

F. Handling Constraint Violations

When soft constraints are violated (multiple parents, cyclic dependencies), SAOA marks affected nodes for manual review rather than producing invalid layouts. This semi-automatic approach acknowledges that real power grids often exhibit

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

topological complexities that require human judgment to resolve optimally.

IV. The Rollwire Layouting Algorithm

A. Problem Setting and Notation

Consider two rectangular symbols A and B placed on an integer grid, with one *docking port* (cable connection point) on each symbol. Let $\boldsymbol{c}_A, \boldsymbol{c}_B \in \mathbb{R}^2$ be the centers of the bounding boxes of A and B, and $\boldsymbol{p}_A, \boldsymbol{p}_B \in \mathbb{R}^2$ be the positions of their docking ports. Docking ports are constrained to lie on the midpoints of the four edges of the bounding box (top, bottom, left, right). The aim of Rollwire is to compute a polyline:

$$(\boldsymbol{h}_0, \boldsymbol{h}_1, \ldots, \boldsymbol{h}_k), \quad \boldsymbol{h}_0 = \boldsymbol{p}_A,\, \boldsymbol{h}_k = \boldsymbol{p}_B \ , \tag{6}$$

such that all segments $\boldsymbol{h}_i\boldsymbol{h}_{i+1}$ are axis-aligned (orthogonal layout) and do not intersect the interior of any symbol.

Define the *port direction vectors*:

$$\boldsymbol{d}_A = \boldsymbol{p}_A - \boldsymbol{c}_A, \qquad \boldsymbol{d}_B = \boldsymbol{p}_B - \boldsymbol{c}_B \ , \tag{7}$$

which encode on which side of the symbol the port is located. For axis-aligned rectangular symbols with docking ports at edge midpoints, the port direction vectors are axis-aligned: each $\boldsymbol{d}_A$ and $\boldsymbol{d}_B$ points in one of the four cardinal directions (right, left, up, or down). We can therefore normalize them to canonical form $\boldsymbol{d}_A, \boldsymbol{d}_B \in (1,0), (-1,0), (0,1), (0,-1)$.

Let $\boldsymbol{\Delta} = \boldsymbol{p}_B - \boldsymbol{p}_A$ be the vector from the start port to the target port. For convenience, define unit direction vectors

$$\hat{\boldsymbol{e}}_x = (1,0), \quad \hat{\boldsymbol{e}}_y = (0,1), \tag{8}$$

and for any non-zero vector $\boldsymbol{v}$, let $\widehat{\boldsymbol{v}} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}$ denote its normalized direction, and $\boldsymbol{v}_\perp = (-v_y,\, v_x)$ a 90-degree rotation used for orthogonal detours.

B. Overlap Detection

Before routing, Rollwire tests whether leaving the source or target port directly in the direction of the other port would cause the connecting segment to cross the corresponding symbol.

For a given port direction vector $\boldsymbol{d} \in \{\boldsymbol{d}_A, \boldsymbol{d}_B\}$, RollWire computes

$$s = \|\boldsymbol{\Delta} + \boldsymbol{d}\|, \qquad t = \|\boldsymbol{\Delta}\| \ . \tag{9}$$

If $s < t$, the straight segment in direction $\boldsymbol{\Delta}$ is considered to overlap the symbol, because shifting the target by $\boldsymbol{d}$ would result in a strictly shorter connecting vector, indicating that the symbol protrudes into the straight-line path. This test is applied separately for the source and the target port.

If neither test indicates an overlap, Rollwire simply draws one straight orthogonal polyline using at most one bend (depending on whether a purely horizontal or vertical alignment is possible).

If at least one overlap is detected, Rollwire introduces an intermediate orthogonal detour as described below.

Input : Start port $\mathbf{p}_S$, target port $\boldsymbol{p}_T$, start port direction vector $\boldsymbol{d}_S$, target port direction vector $\boldsymbol{d}_T$
Output : Ordered list of waypoints $(\boldsymbol{h}_0, \ldots, \boldsymbol{h}_k)$ describing the orthogonal polyline

$\boldsymbol{h}_0 \leftarrow \boldsymbol{p}_S$, $\boldsymbol{\Delta} \leftarrow \boldsymbol{p}_T - \boldsymbol{p}_S$
// Check for overlap conditions:
$o_S \leftarrow (\|\boldsymbol{\Delta} + \boldsymbol{d}_S\| < \|\boldsymbol{\Delta}\|)$, $o_T \leftarrow (\|\boldsymbol{\Delta} + \boldsymbol{d}_T\| < \|\boldsymbol{\Delta}\|)$
if $\neg o_S \wedge \neg o_T$ **then**
  // No overlap: route directly with at most one bend
  polyline $\leftarrow$ DirectRoute$(\boldsymbol{h}_0, \boldsymbol{p}_T)$
  return *polyline*
end
// Overlap detected:
$\hat{\boldsymbol{o}} \leftarrow \widehat{(\boldsymbol{d}_S)_\perp}$, $\boldsymbol{h}_1 \leftarrow \boldsymbol{h}_0 + \hat{\boldsymbol{o}}$
$\boldsymbol{\Delta}' \leftarrow \boldsymbol{p}_T - \boldsymbol{h}_1$
if $|\Delta'_x| \geq |\Delta'_y|$ **then**
  $\boldsymbol{h}_2 \leftarrow \boldsymbol{h}_1 + (\Delta'_x, 0)$
end
else
  $\boldsymbol{h}_2 \leftarrow \boldsymbol{h}_1 + (0, \Delta'_y)$
end
$\boldsymbol{h}_3 \leftarrow \boldsymbol{p}_T$
return $(\boldsymbol{h}_0, \boldsymbol{h}_1, \boldsymbol{h}_2, \boldsymbol{h}_3)$

Figure 3. RollWire Orthogonal Connection Routing

C. Two-Stage Orthogonal Detour

Let $\boldsymbol{p}_S$ denote the start docking port (without loss of generality, the port on A) and $\boldsymbol{p}_T$ the target docking port on B. Let $\mathbf{d}_S$ be the direction vector of the start port.

Rollwire first moves one grid unit in a direction orthogonal to $\boldsymbol{d}_S$ to "escape" the symbol's bounding box:

$$\hat{\mathbf{o}} = \widehat{\mathbf{d}_{S,\perp}}, \qquad \boldsymbol{h}_1 = \boldsymbol{p}_S + \hat{\boldsymbol{o}} \ . \tag{10}$$

This creates a short horizontal (or vertical) segment that clears the start symbol. From $\boldsymbol{h}_1$, Rollwire connects towards $\mathbf{p}_T$ along the dominant component of the remaining difference vector, $\boldsymbol{\Delta}' = \boldsymbol{p}_T - \boldsymbol{h}_1$.

If $|\Delta'_x| \geq |\Delta'_y|$, it draws a horizontal segment, $\boldsymbol{h}_2 = \boldsymbol{h}_1 + (\Delta'_x, 0)$, else a vertical segment, $\boldsymbol{h}_2 = \boldsymbol{h}_1 + (0, \Delta'_y)$.

Finally, RollWire draws one last orthogonal segment from $\boldsymbol{h}_2$ to the target port, $\boldsymbol{h}_3 = \boldsymbol{p}_T$, yielding a three-segment polyline $(\boldsymbol{h}_0, \boldsymbol{h}_1, \boldsymbol{h}_2, \boldsymbol{h}_3)$ with at most two bends. This construction guarantees that the polyline stays outside both rectangular symbols as long as the starting orthogonal step clears the bounding box by at least one grid unit.

D. Algorithmic Formulation

The core routing can be expressed as the following procedure.

Because SAOA guarantees that no other symbols lie between a parent and its child, Rollwire only needs to avoid the two endpoint symbols. This allows a very simple routing scheme with constant-time complexity per connection.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

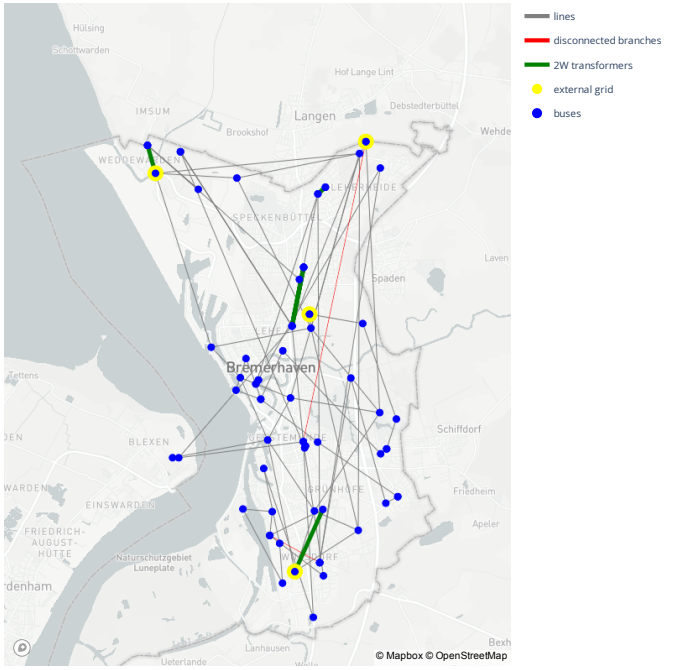

Figure 4. Topographic map of the power grid of the city of Bremerhaven, Germany

E. Segment Width and Endpoint Adjustment

In the implementation, each polyline segment is rendered as a narrow rectangle with width b. Naively joining such rectangles at right angles leads to small overlaps near bend points. To avoid this, Rollwire applies a local correction at each junction between two consecutive segments s_i and s_{i+1}: Let $\hat{\mathbf{u}}_i$ and $\hat{\mathbf{u}}_{i+1}$ be the unit direction vectors of s_i and s_{i+1}. The shared endpoint is shifted by:

$$\frac{b}{2}\,\hat{\mathbf{u}}_i - \frac{b}{2}\,\hat{\mathbf{u}}_{i+1}\ , \tag{11}$$

i. e., by half the width in the direction of the previous segment and by half the width opposite to the next segment. This adjustment ensures that the filled rectangles of adjacent segments meet cleanly without overlapping, producing visually consistent orthogonal connections even when line widths are non-zero.

F. Special Case: Junction Nodes

At junctions (busbars, switchgear symbols) multiple connections may terminate in the same symbol. In these cases, overlaps of line segments within the symbol are visually acceptable because they are covered by the symbol's icon. Rollwire therefore permits multiple connections to share the same central docking point within such symbols, simplifying the routing and keeping the focus on avoiding overlaps outside the node area.

V. Discussion

To illustrate the algorithm, we use a medium-voltage city power grid. Our model represents the power grid of Bremerhaven, Germany. It is part of the MIDAS scenario package [17, 18]. Figure 4 shows the topographic plot of the grid. The data is given to SAOA and Rollwire, which produce the output seen in Figure 5. Note that we show a cutout of the grid due to paper size limits.

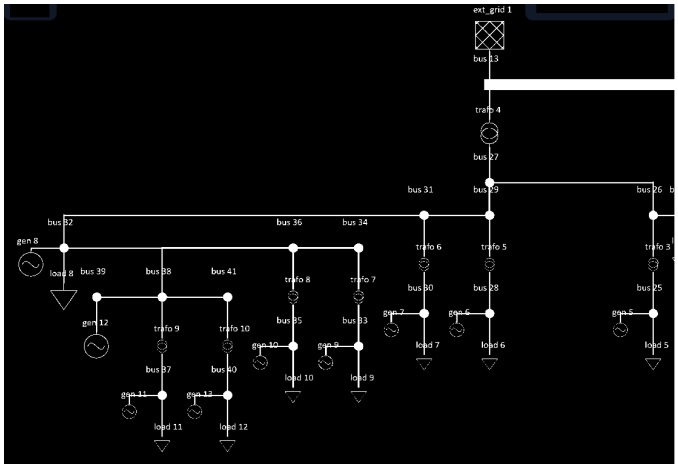

Figure 5. Rollwire's representation of the Bremerhaven power grid

Having established that our algorithm produces acceptable single-line diagrams, we discuss the runtime complexity of the two parts. First, we analyze the runtime complexity of SAOA by examining the following theorem:

Theorem V.1. *SAOA terminates in at most $h+1$ iterations, where $h = \max_{v \in V} d(v)$ is the maximum hierarchical depth of the graph.*

Proof. In each iteration, space requirements propagate upward by one level in the hierarchy. Since changes at depth d can only affect ancestors at depths $< d$, and the root has depth 0, convergence is guaranteed within $h+1$ iterations. □

We can then turn to the drawing mechanism, Rollwire. For a single connection between two ports with direction vectors, Rollwire requires:

- $\mathcal{O}(1)$ vector arithmetic operations for overlap detection, (cf. Eq. (9)).
- $\mathcal{O}(1)$ operations for polyline construction (Algorithm 3).
- $\mathcal{O}(1)$ endpoint adjustments per segment.

For a power grid with m edges (connections), the total routing time is $\mathcal{O}(m)$. Combined with SAOA's $\mathcal{O}(n^{1.5})$ node placement, the overall layout pipeline achieves $\mathcal{O}(n^{1.5}+m)$ complexity, which is practical for typical grids.

VI. Conclusion and Future Work

In this paper, we presented a two-stage algorithm for drawing single line diagrams. Its major contribution is that it produces pleasant, hierarchical layouts that respect voltage levels and are thus suitable to display larger grids with multiple levels.

In the future, we will showcase the algorithm in live demonstrations.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

ACKNOWLEDGEMENTS

SAOA und Rollwire are part of the research project ProRES, funded by the German Federal Ministry for Economic Affairs and Energy under grant no. 03EI4117A.

REFERENCES

[1] M. T. Fischer and D. A. Keim, *Towards a survey of visualization methods for power grids*, Apr. 2022. DOI: 10.48550/arXiv.2106.04661. arXiv: 2106.04661[cs]. Accessed: Jan. 29, 2026. [Online]. Available: http://arxiv.org/abs/2106.04661.

[2] R. Tamassia, "On embedding a graph in the grid with the minimum number of bends," *SIAM Journal on Computing*, vol. 16, no. 3, pp. 421–444, Jun. 1987, Publisher: Society for Industrial and Applied Mathematics, ISSN: 0097-5397. DOI: 10.1137/0216030. Accessed: Jan. 29, 2026. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/0216030.

[3] U. Afzal et al., "Investigating cognitive load in energy network control rooms: Recommendations for future designs," *Frontiers in Psychology*, vol. 13, Mar. 2022, Publisher: Frontiers, ISSN: 1664-1078. DOI: 10.3389/fpsyg.2022.812677. Accessed: Jan. 29, 2026. [Online]. Available: https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2022.812677/full.

[4] E. M. Veith, T. Logemann, A. Wellßow, and S. Balduin, "Play with me: Towards explaining the benefits of autocurriculum training of learning agents," in *2024 IEEE PES Innovative Smart Grid Technologies Europe (ISGT EUROPE)*, Dubrovnik, Croatia: IEEE, 2024, pp. 1–5. DOI: 10.1109/ISGTEUROPE56780.2023.10408277.

[5] T. Biedl and G. Kant, "A better heuristic for orthogonal graph drawings," *Computational Geometry*, vol. 9, no. 3, pp. 159–180, Feb. 1998, ISSN: 0925-7721. DOI: 10.1016/S0925-7721(97)00026-6. Accessed: Jan. 29, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925772197000266.

[6] K. Freivalds and J. Glagoļevs, "Graph compact orthogonal layout algorithm," in *Combinatorial Optimization*, P. Fouilhoux, L. E. N. Gouveia, A. R. Mahjoub, and V. T. Paschos, Eds., Cham: Springer International Publishing, 2014, pp. 255–266, ISBN: 978-3-319-09174-7. DOI: 10.1007/978-3-319-09174-7_22.

[7] S. Kieffer, T. Dwyer, K. Marriott, and M. Wybrow, "HOLA: Human-like orthogonal network layout," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 349–358, 2015. DOI: 10.1109/TVCG.2015.2467451.

[8] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for visual understanding of hierarchical system structures," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 2, pp. 109–125, 1981. DOI: 10.1109/TSMC.1981.4308636.

[9] U. Brandes and B. Köpf, "Fast and simple horizontal coordinate assignment," in *Graph Drawing*, ser. Lecture Notes in Computer Science, vol. 2265, Springer, 2002, pp. 31–44. DOI: 10.1007/3-540-45848-4_3.

[10] A. B. Birchfield and T. J. Overbye, "Techniques for drawing geographic one-line diagrams: Substation spacing and line routing," *IEEE Transactions on Power Systems*, vol. 33, no. 6, pp. 7269–7277, 2018. DOI: 10.1109/TPWRS.2018.2846283.

[11] N. Raman and R. K. Schwartzenberg, "Automatic generation of power system one-line diagrams," in *IFAC Proceedings Volumes*, vol. 19, 1986, pp. 231–236. DOI: 10.1016/S1474-6670(17)59383-0.

[12] I. Lendák and A. Poth, "Electric power system one-line diagram generation technology," in *Proceedings of the IEEE International Energy Conference*, 2012, pp. 152–157. DOI: 10.1109/EnergyCon.2012.6348286.

[13] S. P. S. Hong, J. Y. Lee, and P. S. Mah, "Power system connectivity visualization using an orthogonal graph layout algorithm," *Journal of Computing Science and Engineering*, vol. 16, no. 4, pp. 187–199, 2022. DOI: 10.5626/JCSE.2022.16.4.187.

[14] C. Yang, J. Wang, and D. Shi, "Automatic layout of one-line diagrams for large transmission systems based on graph partitioning," in *2024 IEEE Power & Energy Society General Meeting (PESGM)*, ISSN: 1944-9933, Jul. 2024, pp. 1–5. DOI: 10.1109/PESGM51994.2024.10689189. Accessed: Jan. 29, 2026. [Online]. Available: https://ieeexplore.ieee.org/document/10689189.

[15] P. Sen et al., "An Automatic Generation Method for Single-line Diagram of Distribution Network With Supporting Incremental Update," *Modern Electric Power*, vol. 41, no. 1, pp. 21–28, Feb. 2024, ISSN: 1007-2322. DOI: 10.19725/j.cnki.1007-2322.2022.0189. Accessed: Jan. 29, 2026. [Online]. Available: http://xddl.ncepujournal.com/en/article/doi/10.19725/j.cnki.1007-2322.2022.0189.pdf.

[16] P. S. Nagendra Rao and R. Deekshit, "Distribution feeder one-line diagram generation: A visibility representation," *Electric Power Systems Research*, vol. 70, no. 3, pp. 173–178, Aug. 2004, ISSN: 0378-7796. DOI: 10.1016/j.epsr.2003.12.005. Accessed: Jan. 29, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378779603003031.

[17] E. Veith et al., "palaestrAI: A training ground for autonomous agents," in *Proceedings of the 37th annual European Simulation and Modelling Conference*, Oct. 2023.

[18] S. Balduin, E. M. S. P. Veith, and S. Lehnhoff, "MIDAS: An open-source framework for simulation-based analysis of energy systems," in *Simulation and Modeling Methodologies, Technologies and Applications*, G. Wagner, F. Werner, and F. De Rango, Eds., Cham: Springer International Publishing, 2023, pp. 177–194, ISBN: 978-3-031-43824-0.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org