



MASTER'S DEGREE IN INDUSTRIAL ENGINEERING

MASTER'S THESIS

Automatic Single-Line Diagram Generation for LV Networks from GIS Data

Author: Pablo Bermejo Villaescusa

Supervisor: Stuart Galloway

Co-Supervisor: Connor McGarry

Industry Supervisor: Ciaran Higgins

Madrid

August 2025

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título
Automatic Single-Line Diagram Generation for LV Networks from GIS Data
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el
curso académico 2024/2025 es de mi autoría, original e inédito y
no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido
tomada de otros documentos está debidamente referenciada.

Fdo.: Pablo Bermejo Villaescusa Fecha: 15/08/2025



Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Stuart Galloway Fecha: 15/08/2025



Fdo.: Connor McGarry Fecha: 15/08/2025



Fdo.: Ciaran Higgins Fecha: 15/08/2025





MÁSTER EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

Generación Automática de Diagramas Unifilares para Redes de Baja Tensión a partir de Datos de GIS

Autor: Pablo Bermejo Villaescusa

Director: Stuart Galloway

Co-Director: Connor McGarry

Director Industrial: Ciaran Higgins

Madrid

Agosto de 2025

Acknowledgements

I am deeply grateful to the many people whose support have made this project possible.

First, I owe my most sincere gratitude to my parents, for the education, values and encouragement to always follow the path I wanted.

I am also profoundly grateful to my supervisors, Stuart, Connor and Ciaran, for their guidance, thoughtful feedback, and generous availability, which have made this project a challenging yet very rewarding experience.

My heartfelt thanks also go to Blanca, whose patience and steady support have sustained me in the most demanding periods, and without it I would not have been able to complete this project.

Finally, I also wish to thank everyone who has shaped my studies from my bachelor's degree through this master's, including mentors, and friends. I am grateful to each of you for making this journey so rich and memorable.

AUTOMATIC SINGLE-LINE DIAGRAM GENERATION FOR LV NETWORKS FROM GIS DATA

Author: Bermejo Villaescusa, Pablo

Supervisor: Galloway, Stuart.

Collaborating Entity: ScottishPower Energy Networks

Abstract

Introduction

Low-voltage (LV) distribution networks are changing rapidly with the increasing penetration of distributed energy resources (DERs), including rooftop PV, distribution-level storage, and EVs, turning formerly passive feeders into complex active, bi-directional systems [1]. Utilities already maintain rich network models in geographic information systems (GIS), yet raw geospatial representations do not conveniently convey connectivity and topology at a glance. For this purpose, single-line diagrams (SLDs) remain the most intelligible tool, but manual drafting does not scale well to utility-sized datasets. This thesis addresses that gap by automating the generation of standard, orthogonal SLDs directly from heterogeneous GIS models.

Prior work on automatic diagram generation either focuses on idealized radial trees, general graph drawings that ignore SLD conventions, or proprietary pipelines. There is currently no end-to-end available solution that, taking networks of variable complexity as an input, can automatically generate a single-line diagram of the underlying network. The objective of the thesis is, therefore, to establish an end-to-end automated pipeline that: (i) represents connectivity and key network information (keeping elements like switches, fuses, transformers), (ii) minimizes clutter by creating groups of consumers and simplifying network sections, and (iii) runs fast enough for interactive or near-interactive use.

Project definition and methodology

The thesis frames SLD generation as a graph transformation and visualization problem. It builds a graph equivalent of the underlying network, applies a simplification algorithm and computes a layered orthogonal graph layout through an adaptation of the Sugiyama framework [2], a widely adopted approach to generate layered graph layouts through the following phases: cycle removal, layer assignment, crossing minimization, and coordinate assignment. The crossing minimization phase is NP-hard, so a heuristic approach is necessary.

The methodology consists in a 4-phase modular pipeline which converts raw GIS data to orthogonal single line diagrams that represent the underlying topology of the network. The four phases are the following:

1. **Graph classification & simplification.** Applies a set of classification rules to the nodes in the graph guided by input adjustable parameters that allow to determine how the graph is simplified. The simplification algorithm then creates groups of consumers and adjacent nodes that do not convey topological information, while ensuring that key elements are retained in the final schematic (e.g. transformers, switches or fuses).
2. **Normalization.** Enforces a standard node structure in the underlying graph that is compatible with the later layout and plotting algorithms.
3. **Layered layout.** Algorithm built on the Sugiyama approach to generate a layered layout; employs a fixed layer assignment and iterative crossing minimization (barycenter/median) with randomized restarts to escape local minima. After the crossing minimization, nodes in the graph are assigned discrete grid positions to meet the orthogonality requirements.
4. **Plotting & export.** Assigns standard single-line symbols based on the positions calculated in the layout and exports in SVG format.

This modular pipeline design allows individual stages to be modified or replaced while preserving a working end to end solution. This ensures that the solution can be fitted to data

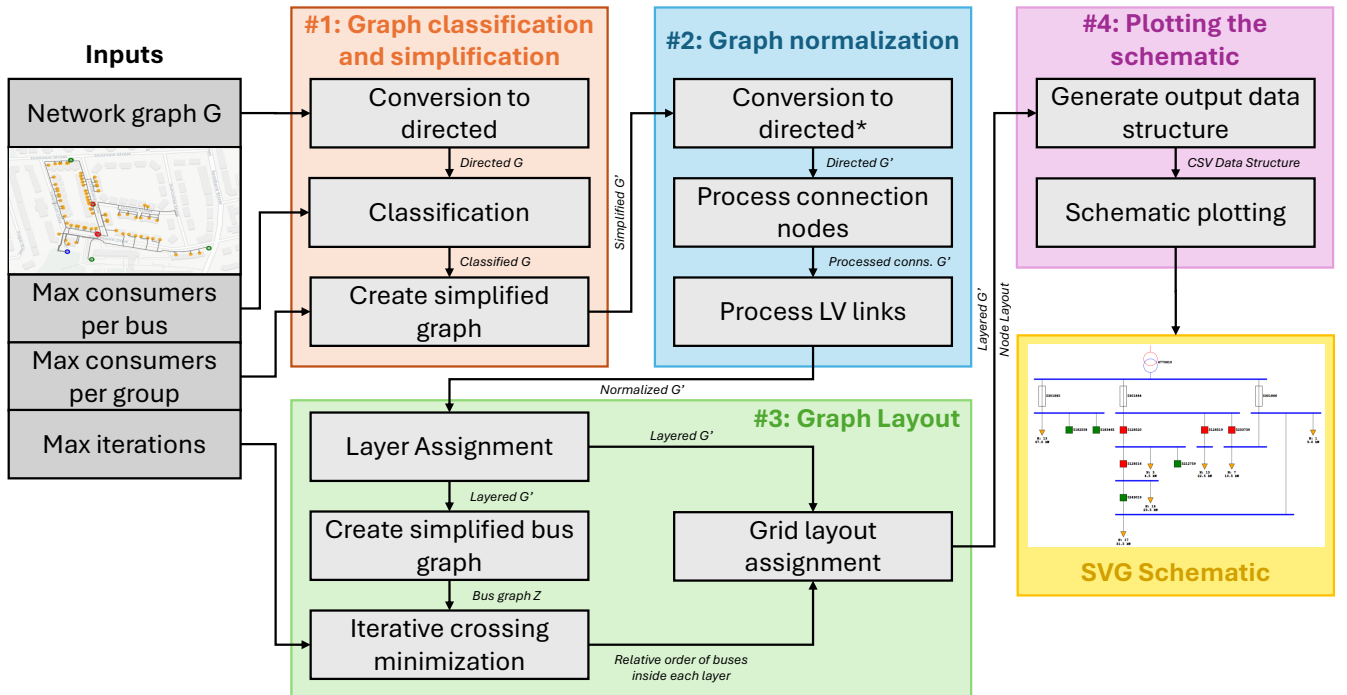


Figure A: Methodology diagram.

with different characteristics and to produce schematics with different styles or format. A diagram describing the process can be observed in Figure A.

Results

The pipeline has been tested on eight different networks of *Scottish Power Energy Networks* distribution grid data, with significantly different characteristics, both in size, measured by the number of nodes, and meshedness, measured by the number of nodes with two predecessors in the equivalent graph. The input set of parameters was kept constant across all tests to ensure comparability among results, and several metrics were measured to benchmark the algorithms performance. These metrics include algorithm runtime (in seconds), k the number of crossings in the final layout, N_{iter} , the number of iterations of the naïve layout algorithm it takes to find the solution with least crossings, and N_L , the number of nodes in the final layout. The results of the case study can be observed in Figure B.

	Input		Results			
	N_{NODES}	M	k	N_{iter}	N_L	Runtime (s)
Case Study 1	69	0	0	1	13	0.1
Case Study 2	145	1	0	1	15	0.13
Case Study 3	239	2	0	1	36	0.58
Case Study 4	611	2	0	1	67	1.57
Case Study 5	1126	5	0	4	64	5.07
Case Study 6	1233	4	2	20	90	6.43
Case Study 7	2282	11	8	77	127	21.18
Case Study 8	1810	7	0	27	643	17.87

Figure B: Summary of case studies results.

The resulting schematics were considered valid only if the underlying graph layout had no crossings, as the existence of these crossings causes elements to overlap; generating invalid connectivity information and therefore making the schematics unsuitable for operational use. The results show that the algorithm generated valid schematics for 6 of the 8 networks tested, and for the two that it did not, the fixed layer assignment, which caused non-level-planar structures in the underlying graph, that the crossing minimization algorithm was not able to resolve. Runtime was only ~0.1 to 1.6 seconds for the simpler networks, but it increased significantly for the more complex ones. Figure C shows the resulting single line diagram from one of the case studies.

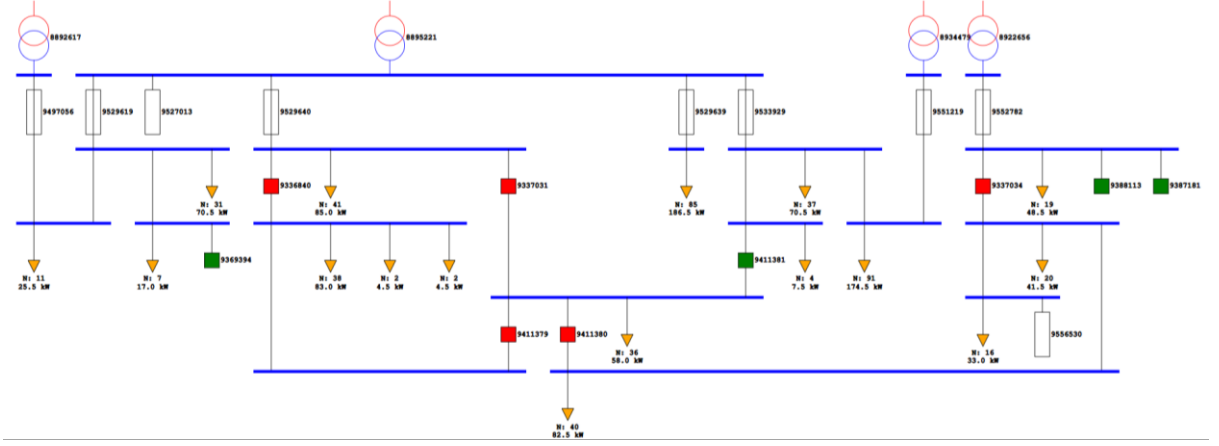


Figure C: Example output single line diagram

Additionally, the results include an analysis of how tuning input parameters affects the characteristics of the final schematic and whether those effects match expectations. The layout algorithm’s ability to minimize edge crossings in complex networks was also benchmarked by progressively increasing the number of maximum naive iterations and measuring the resulting runtime growth. The findings show that the parameters offer useful, though not perfectly precise, control over the output, and that additional naïve iterations of the layout algorithm were effective in helping the algorithm help escape local optima and reach lower crossing solutions, at the cost of higher computational effort.

Conclusions

This project delivers a reproducible pipeline to translate LV distribution grid data into standard, readable single line diagrams while preserving the network’s topology and its key elements. By separating graph simplification, layout and schematic plotting, the approach is modular and tunable via input parameters to match desired diagram detail. Applied to eight diverse real networks, it produced six valid diagrams, and the two failures aligned with level-planarity violations caused by the fixed layer assignment. Future potential improvements to the pipeline include solving the level-planarity problem through either a joint MILP formulation to co-optimize layers and horizontal layout, or another heuristic approach to layer assignment; and optimizing parts of the process so that runtime does not scale as significantly with network complexity.

GENERACIÓN AUTOMÁTICA DE DIAGRAMAS UNIFILARES PARA REDES DE BAJA TENSIÓN A PARTIR DE DATOS DE GIS

Autor: Bermejo Villaescusa, Pablo.

Director: Galloway, Stuart.

Entidad Colaboradora: ScottishPower Energy Networks

Resumen del Proyecto

Introducción

Las redes de distribución de baja tensión (LV) están cambiando rápidamente con la creciente penetración de recursos energéticos distribuidos (DERs), incluyendo generación distribuida, almacenamiento a nivel de distribución y vehículos eléctricos, convirtiendo redes anteriormente pasivas en sistemas activos y bidireccionales [1]. Los operadores de la red ya mantienen modelos de red detallados en sistemas de información geográfica (GIS), pero las representaciones geoespaciales en bruto no transmiten de manera eficaz la conectividad y la topología rápidamente. Con este fin, los diagramas unifilares siguen siendo la herramienta más inteligible, pero la elaboración manual no escala bien a conjuntos de datos del tamaño de una red de distribución. Este proyecto aborda esa brecha, automatizando la generación de diagramas ortogonales y estándar directamente a partir de modelos de GIS con diferentes características.

Actualmente no existe una solución completa disponible que, partiendo de datos sobre redes de variable complejidad, pueda generar automáticamente diagramas unifilares de la red subyacente. El objetivo de la tesis es, por tanto, establecer un proceso automático completo de extremo a extremo que: (i) represente la conectividad y la información clave de la red (manteniendo elementos como interruptores, fusibles y transformadores), (ii) simplifique la información creando grupos de consumidores, y (iii) funcione con suficiente rapidez para implementarse a nivel interactivo o casi interactivo.

Definición del proyecto y metodología

Este proyecto aborda la generación de diagramas como un problema de transformación y visualización de grafos. En primer lugar, se construye un grafo equivalente de la red subyacente, sobre el que se aplica un algoritmo de simplificación y se calcula un dibujo del grafo ortogonal por capas mediante una adaptación del método de Sugiyama [2], un enfoque ampliamente utilizado para generar trazados de grafos por niveles a través de las siguientes fases: eliminación de ciclos, asignación de capas, minimización de cruces y asignación de coordenadas. Dado que la minimización de cruces es NP-completa, se recurre a un enfoque heurístico.

La metodología consiste en un proceso modular de cuatro fases que transforma datos GIS en bruto en diagramas unifilares ortogonales que representan la topología subyacente de la red. Las cuatro fases son:

1. **Clasificación y simplificación del grafo.** Se aplica un conjunto de reglas de clasificación a los nodos del grafo, guiado por parámetros de entrada configurables que determinan cómo se simplifica. El algoritmo agrupa consumidores y nodos adyacentes que no aportan información topológica, garantizando que los elementos clave (por ejemplo, transformadores, interruptores o fusibles) se conserven en el esquema final.
2. **Normalización.** Impone en el grafo una estructura de nodos estándar compatible con los algoritmos posteriores de diseño y trazado.
3. **Dibujo por capas.** Algoritmo basado en el método de Sugiyama para generar un diseño por capas; emplea una asignación fija de capas y una minimización iterativa de cruces con reinicios aleatorios para escapar de óptimos locales. Tras la minimización, se asignan a los nodos posiciones discretas, equivalente a una disposición en rejilla para cumplir el requisito de ortogonalidad.
4. **Trazado y exportación.** Asigna símbolos unifilares estándar en función de las posiciones calculadas y exporta el resultado en formato SVG.

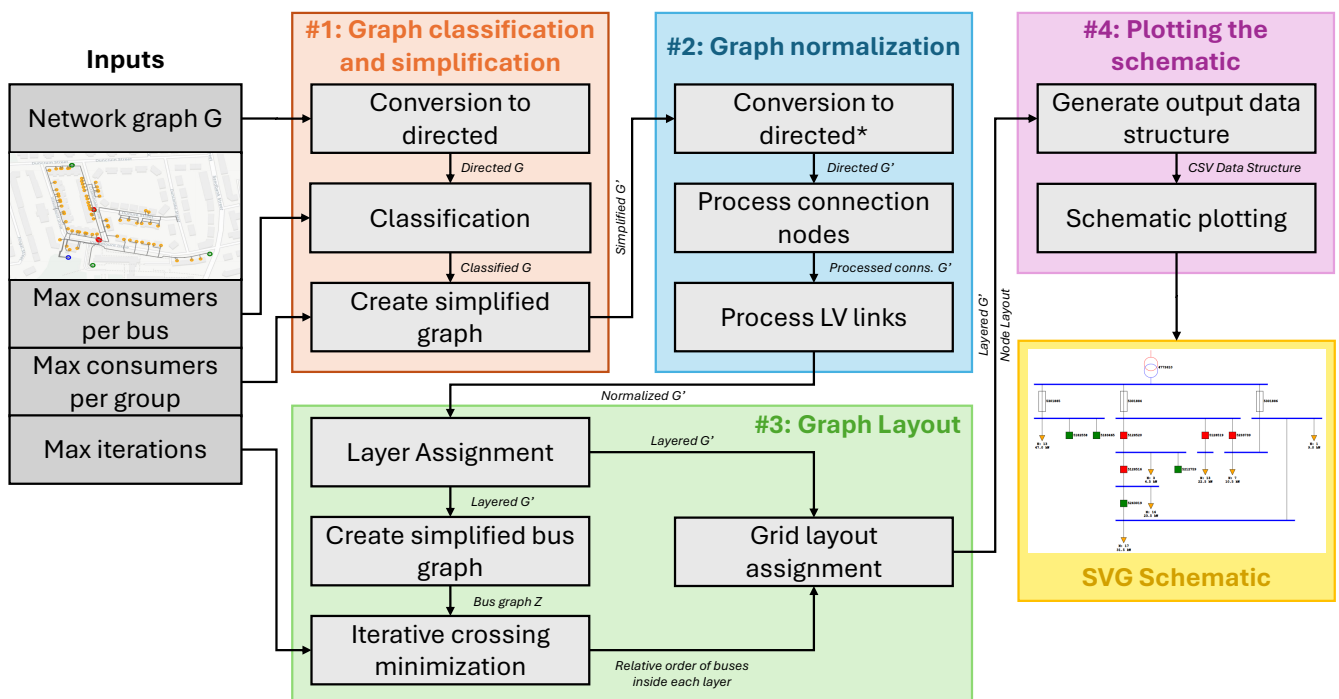


Figura A: Esquema de la metodología

Este enfoque modular permite modificar o sustituir partes individuales manteniendo una solución funcional de extremo a extremo, lo que facilita la adaptación a datos con distintas características y la producción de esquemas con diferentes estilos o formatos. La Figura A ilustra el proceso.

Resultados

El algoritmo se ha evaluado en ocho secciones diferentes de la red de distribución de *ScottishPower Energy Networks*, con características significativamente distintas, tanto en tamaño, medido por el número de nodos, como en el grado de mallado, M , medido por el número de nodos con dos predecesores en el grafo equivalente. El conjunto de parámetros de entrada se mantuvo constante en todas las pruebas para garantizar la comparabilidad entre resultados, y se midieron varias métricas para evaluar el rendimiento del algoritmo. Estas métricas incluyen el tiempo de ejecución del algoritmo (en segundos); k , el número de cruces en el diseño final; N_{iter} , el número de iteraciones aleatorias del algoritmo de reducción de cruces que se requieren para encontrar la solución con menos cruces; y N_L , el número de nodos en el diseño final. Los resultados del estudio pueden observarse en la Figura B.

	Input		Results			
	N_{NODES}	M	k	N_{iter}	N_L	Runtime (s)
Case Study 1	69	0	0	1	13	0.1
Case Study 2	145	1	0	1	15	0.13
Case Study 3	239	2	0	1	36	0.58
Case Study 4	611	2	0	1	67	1.57
Case Study 5	1126	5	0	4	64	5.07
Case Study 6	1233	4	2	20	90	6.43
Case Study 7	2282	11	8	77	127	21.18
Case Study 8	1810	7	0	27	643	17.87

Figura B: Resultados de los casos de estudio

Los esquemas resultantes se consideraron válidos solo si el grafo subyacente no presentaba cruces, ya que esto provoca solapamientos de elementos en el diagrama, generando información de conectividad inválida y, por tanto, causando que los diagramas no sean aptos para uso operativo. Los resultados muestran que el algoritmo generó esquemas válidos para 6 de las 8 redes probadas, y que en las dos en las que no lo hizo, la asignación fija de capas causó estructuras no planares por niveles en el grafo subyacente que el algoritmo de minimización de cruces no fue capaz de resolver. El tiempo de ejecución fue de $\sim 0,1$ a 1,6 segundos para las

redes más simples, pero aumentó significativamente para las más complejas. La Figura C muestra el diagrama unifilar resultante para una de las redes estudiadas.

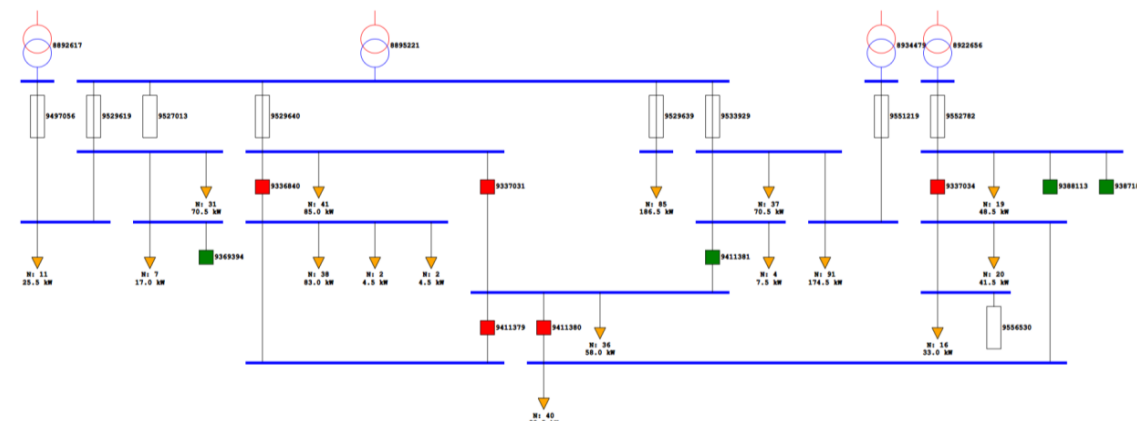


Figura C: Ejemplo de diagrama resultante del proceso

Adicionalmente, los resultados incluyen un análisis de cómo el ajuste de los parámetros de entrada afecta a las características del esquema final, y si esos efectos se alinean con las expectativas. La capacidad del algoritmo de asignar posiciones para minimizar cruces en redes complejas también se evaluó, aumentando progresivamente el número máximo de iteraciones aleatorias y midiendo el aumento resultante del tiempo de ejecución, y el número de cruces en la mejor solución encontrada. Los resultados muestran que los parámetros ofrecen un control útil, aunque no perfectamente preciso, sobre el resultado, y que iteraciones aleatorias adicionales del algoritmo de minimización de cruces son eficaces para ayudar al algoritmo a escapar de óptimos locales y alcanzar soluciones con menos cruces, a costa de un mayor esfuerzo computacional.

Conclusiones

El proyecto propone un proceso reproducible para traducir datos de redes de distribución a diagramas unifilares estándar y fácilmente legibles, preservando la topología de la red y sus elementos clave. El enfoque es modular y el resultado configurable mediante parámetros de entrada para ajustarse al nivel de detalle deseado del diagrama. Aplicado a ocho redes reales diversas, produjo seis diagramas válidos; los dos fallos son causados por la asignación fija de capas en el grafo subyacente. Posibles mejoras futuras del proceso incluyen sustituir la asignación fija de capas por una formulación MILP conjunta para co-optimizar la asignación de capas y la reducción de cruces, u otro enfoque heurístico para la asignación de capas; así como optimizaciones de partes del proceso para que el tiempo de ejecución no escale de forma exponencial con la complejidad de la red.

Table of Contents

Acknowledgements	v
Abstract	vi
Resumen del Proyecto	x
1 Introduction	1
1.1 Research Context and Motivation	1
1.2 Problem Statement and Objectives.....	1
1.3 Thesis Structure	2
2 Background and Literature Review	3
2.1 DER Integration and Challenges in LV Networks.....	3
2.1.1 Technical challenges on LV networks	3
2.1.2 Need for observability and monitoring.....	4
2.2 Visualization of Power System Data	6
2.2.1 Evolution of visualization systems in power grids.....	6
2.2.2 Geographical vs topological references.....	7
2.2.3 Overview of current approaches to display grid data	8
2.2.4 Innovative approaches	13
2.3 Modelling LV Networks as Graphs	14
2.3.1 Applications of Graph Theory in Power Systems	15
2.3.2 Advantages of Graph-Based Modelling	16
2.4 Geographic Information Systems in the electric power sector.....	18
2.4.1 GIS in electric distribution networks.....	18
2.4.2 Key applications of GIS in power utilities	20
2.4.3 GIS as a foundation for grid modernization	20
2.5 Automatic Schematic Generation of Single Line Diagrams	22
2.5.1 Evolution of automatic diagram generation	22
2.5.2 Generation of single line diagrams.....	23

2.5.3	Commercial landscape and industrial implementations	26
2.5.4	Current research gap	27
2.6	Summary.....	29
3	Foundations of Graph Theory	30
3.1	Fundamentals and Core Concepts	30
3.1.1	Basic concepts	30
3.1.2	Graph representations	32
3.1.3	Graph properties	33
3.1.4	Graph traversal algorithms	35
3.1.5	Directed acyclic graphs	38
3.1.6	Planarity.....	40
3.2	Principles of Graph Layout Algorithms	42
3.2.1	Reingold-Tilford algorithm	42
3.2.2	Hierarchical layouts - The Sugiyama framework.....	47
3.3	Summary.....	52
4	Methodology and Implementation.....	53
4.1	Formal Problem Framing	53
4.2	Solution Overview	56
4.3	Implementation Details.....	60
4.3.1	Development language and dependencies	60
4.3.2	Data characteristics.....	60
4.4	Model Development	63
4.4.1	Graph classification and simplification	64
4.4.2	Graph normalization	71
4.4.3	Graph layout generation	81
4.4.4	Schematic plotting	94
5	Case Studies	96
5.1	Experimental Setup	96

5.1.1	Overview of network characteristics	96
5.1.2	Result validation	97
5.1.3	Input parameters	97
5.1.4	Objectives	98
5.1.5	Hardware:	98
5.2	Evaluation of Case Study Outcomes	99
5.2.1	Case Study 1: Small Radial Network	99
5.2.2	Case Study 2: Small Quasi-Radial Network	102
5.2.3	Case Study 3: Small Meshed Network	105
5.2.4	Case Study 4: Medium Sized Network #1	109
5.2.5	Case Study 5: Medium Sized Network #2	111
5.2.6	Case Study 6: Medium Sized Network #3	113
5.2.7	Case Study 7: Large Network.....	116
5.2.8	Case Study 8: MV Network.....	118
6	Results and Discussion	120
6.1	Summary of Results and Key Findings	120
6.2	Layout Algorithm Performance Analysis	122
6.3	Method Limitations and Design Trade-offs.....	125
7	Conclusions and Future Work	127
8	References	129
9	Appendix I: Alignment with the UN Sustainable Development Goals.....	134

List of Figures

Figure 2-1: Distributed generation share of total installed generation in the UK, 2022 [1].....	3
Figure 2-2: Substation's topological single-line diagram from GE's ADMS [9]	8
Figure 2-3: Line flow visualization of the US transmission system [9].....	9
Figure 2-4: Contour of locational marginal prices in New England [9].....	10
Figure 2-5: Line contouring representation of power transfer distribution factor [9].....	11

Figure 2-6: GDV displaying power reserves in the US [10].	12
Figure 2-7: GreenGrid's output (right) for a section of the WECC grid (left) [11].	12
Figure 2-8: GreenCurve output for a section of the WECC grid, displaying phase angle [12].	13
Figure 2-9: ChatGrid dynamic visualization created through NLP [15].	14
Figure 2-10: (a) GIS view of an LV feeder (b) Corresponding graph representation.	15
Figure 2-11: GIS Representation of a SPEN LV Feeder.....	19
Figure 2-12: (a) Underlying Graph and (b) Equivalent representation proposed by Canales-Ruiz et al.[30].	23
Figure 2-13: (a) Rooted tree layout, and (b) resulting bus structure proposed by Nao et al. [31].	24
Figure 2-14: DIgSILENT's PowerFactory diagram layout [37].....	26
Figure 3-1: Simple 3-node graph.....	30
Figure 3-2: Directed graph with a looped edge.	31
Figure 3-3: Edge list.	32
Figure 3-4: Adjacency (left) and incidence (right) matrices.....	33
Figure 3-5: Directed path between nodes C and B.....	34
Figure 3-6: Unconnected, weakly connected and strongly connected graph (respectively). ...	34
Figure 3-7: BFS algorithm.....	36
Figure 3-8: DFS algorithm.	38
Figure 3-9: DAG and cycle-containing directed graph (respectively).	39
Figure 3-10: FAS of non-acyclical graph.	40
Figure 3-11: Evaluation of topological planarity and reliability for interference reduction in radio sensor networks.	40
Figure 3-12: Radial rooted tree [43].	43
Figure 3-13: Radial rooted tree with mod and x assigned [43].	44
Figure 3-14: Radial rooted tree with mod, x and shift assigned [43].	46
Figure 3-15: Radial rooted tree with the resulting x positions [43].	47

Figure 3-16: Non-level-planar graph.....	48
Figure 4-1: Methodology diagram.....	58
Figure 4-2: Example of an input LV network section.	61
Figure 4-3: Node legend for graph figures	63
Figure 4-4: Classification and simplification phase.	64
Figure 4-5: (a) Classified graph G input (b) Simplified graph G', output of the simplification algorithm.....	70
Figure 4-6: Graph normalization phase.	71
Figure 4-7: Connected LV links normalization (yellow nodes).	76
Figure 4-8: Disconnected adjacent LV link nodes.....	77
Figure 4-9: Single degree LV link nodes with adjacent IDs.....	78
Figure 4-10: Adjacent key nodes, requiring an additional bus node (Represents a distribution transformer and its connected fuses).	79
Figure 4-11: Two scenarios requiring additional bus nodes (left), and the resulting structures after adding the bus nodes (right).	80
Figure 4-12: Scenario that would lead to a single-degree bus node, after processing connection nodes.	80
Figure 4-13: Graph layout phase.	81
Figure 4-14: Node position assignment for elements inside the same bus.....	82
Figure 4-15: Example complete normalized graph G'.....	85
Figure 4-16: Resulting bus simplified graph corresponding to G' in Figure 4.14.....	85
Figure 4-17: Zero crossings layout of the graph in Figure 4.16.....	87
Figure 4-18: Grid layout of the simplified bus graph in Figure 4.15.	90
Figure 4-19: Column gap to allow the placement of bus nodes with two predecessors, in the layout in Figure 4.18.....	91
Figure 4-20: Connection node positions.....	91
Figure 4-21: Final complete graph layout, for the sample graph in Figure 4.15.....	93
Figure 4-22: diagram plot of a length=3 bus node (same Figure as 4.14).....	94

Figure 4-23: Schematic symbols used in the final diagram.	95
Figure 4-24: Final diagram, for the sample graph in Figure 4.15.	96
Figure 5-1: (a) Case study 1 network GIS representation (b) Zoomed in version showing there are three independent feeders branching down from the transformer.	99
Figure 5-2: Case study 1 output schematic.....	99
Figure 5-3: Case study 1 parameter adjustment results.	101
Figure 5-4: Case Study 2 network GIS representation.	102
Figure 5-5: Case study 2 output diagram.....	103
Figure 5-6: Case study 3 network GIS representation.....	105
Figure 5-7: Case study 3 output diagram.....	106
Figure 5-8: Comparison of original network sections and their diagram representation.	108
Figure 5-9: Case study 4 network GIS representation.....	109
Figure 5-10: Case study 4 output diagram.....	109
Figure 5-11: Case study 5 network GIS representation.....	111
Figure 5-12: Case study 5 output diagram.....	111
Figure 5-13: Case study 6 network GIS representation.....	113
Figure 5-14: Case study 6 output diagram.....	113
Figure 5-15: Overlapping buses in the final diagram.	114
Figure 5-16: underlying non-level-planar graph of network 6.	114
Figure 5-17: Case study 7 network GIS representation.....	116
Figure 5-18: Case study 7 output diagram.....	116
Figure 5-19: Underlying non-level-planar graph of network 7.	117
Figure 5-20: Case study 8 network GIS representation.....	118
Figure 5-21: Case Study 8 output diagram.	118
Figure 5-22: Case study 8 output diagram, zoomed-in view.....	119
Figure 6-1: Layout algorithm performance, case studies 6, 7 and 8.	122

List of Tables

Table 4-1: Properties used in the implementation.61

Table 5-1: Case study networks characteristics 96

Table 6-1: Summary of case studies results..... 120

1 Introduction

1.1 Research Context and Motivation

LV distribution networks are experiencing a significant shift due to the growing penetration of distributed energy resources (DERs), including rooftop PV, distribution level storage and electric vehicles, which are turning traditionally passive LV feeders into active bidirectional networks. The resulting behaviors (voltage instability, protection coordination challenges, switching network topology) call for necessary increased observability and the development of tools that help network operators and planning engineers to quickly understand network connectivity, topology, and system status information at a glance.

Distribution system operators (DSOs) typically maintain comprehensive network databases based on geographical information systems (GIS) which contain information about network assets and their location. However, due to their scale and vast amount of data, representations of this raw data do not intuitively provide easily graspable connectivity and topology information.

Single line diagrams are better suited for representing network connectivity and its underlying topology in an easier to process way for operators. However, the traditional approach of manually drafting the single line diagrams is slow and therefore impractical for visualizing distribution networks, which may contain millions of elements. This creates both a need and opportunity, to develop automatic methods to generate single line diagrams leveraging the large GIS databases maintained by DSOs.

1.2 Problem Statement and Objectives

There is currently no open, reproducible end-to-end workflow that converts heterogeneous GIS-based LV network models into standard, orthogonal, single-line diagrams at scale. Existing approaches are either partial or proprietary; academic work often targets either radial trees or generating general graph drawings not focusing on single line diagram conventions. Crossing minimisation on layered, meshed graphs is challenging; under fixed layer rankings, non-level-planar substructures make zero-crossing layouts impossible without re-ranking.

Therefore, the goal of the thesis is to implement a pipeline that generates standardized orthogonal single line diagrams from GIS data, ensuring that the algorithm:

- Preserves topology and elements connectivity information (switches, fuses, transformers, etc.).
- Minimizes visual clutter, grouping consumers and displaying only essential topological information.
- Runs fast enough for interactive or near interactive use on average personal computing hardware.

The thesis is implemented on distribution network GIS data from *Scottish Power Energy Networks (SPEN)*.

1.3 Thesis Structure

The thesis is divided into the following chapters:

- **Chapter 2, Background and literature review.** Surveys DER-caused LV challenges, power-system visualization (geographical, topological and hybrid), graph-based modelling, GIS in electric utilities, and automatic diagram generation.
- **Chapter 3, Foundations of graph theory.** Introduces several graph theory concepts and layout algorithms that are used in the methodology.
- **Chapter 4, Methodology and implementation.** Formalizes the problem, details the pipeline and implementation choices from data input to plotting/export.
- **Chapter 5, Case studies.** Applies the pipeline to eight networks of increasing size and meshing, documents inputs and outputs.
- **Chapter 6, Results and discussion.** Summarizes the case studies results and analyzes the performance of the layout algorithm on complex networks.
- **Chapter 7, Conclusions and future work.** Reflects on contributions and outlines next steps.

2 Background and Literature Review

2.1 DER Integration and Challenges in LV Networks

The drive to decarbonize energy systems, and transition to a renewable-based generation has spiked the development of renewable energy technologies over the past 20 years. Along with the construction of large, centralized solar and wind power plants, the growth of small-scale renewable generation at the distribution level, like rooftop PV and wind, has also contributed significantly to the decarbonization of the energy mix in developed economies. According to Gordon et al [1], by 2022 the installed distributed generation capacity was 35% of the total installed capacity in Great Britain.

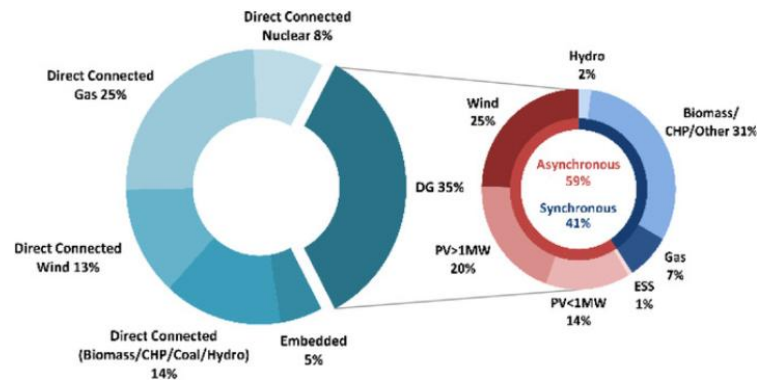


Figure 2-1: Distributed generation share of total installed generation in the UK, 2022 [1].

In addition to distributed generation, the adoption of residential battery storage systems and battery electric vehicles has also grown significantly over the past 20 years. In the United Kingdom, battery electric vehicles comprised 19% of all new car registrations in 2024 [3]. This trend causes distribution grid-level demand to rise significantly, and smart charging and vehicle to grid technologies can potentially cause significant shifts in the demand curve.

2.1.1 Technical challenges on LV networks

This evolution has fundamentally transformed distribution grids, which were initially designed for one-way power flow, from substations to consumers. Traditionally passive radial feeders now experience bidirectional power flows, dynamic voltage fluctuations and more complex protection requirements due to DER power injections. In essence, the distribution system has become an active network, which introduces new operational paradigms and challenges.

One of the major impacts of high DER penetration is the effect it has on voltage stability and power quality. For example, rising voltage in LV feeders during periods of high solar irradiance

has become a major source of concern. When several small-scale PV systems export power into the same sections of LV feeders, the resulting overvoltage can damage customer equipment and cause phase imbalances [4].

Another significant challenge lies in protection coordination and safety. Traditional protection systems rely on the fact that power flows run exclusively in one direction, from the substation to the consumers, and consider fault current magnitudes based on this. High DER penetration however alters these assumptions, reverse currents can feed faults from the far end, blinding upstream protection, and making it impossible to isolate the fault, if proper distributed generation protection systems are not put in place. The magnitude of fault currents provided by distributed generations are comparatively low, so traditional overcurrent protection systems might fail to respond, requiring the review of fuse and relay settings, or introducing bidirectional relays and advanced protection systems, like adaptive or transfer trip relays. Another significant risk is unintended islanding, which may occur when a portion of the grid is electrically isolated, and the DERs in that area keep feeding local loads, creating an island. Preventing and detecting those islands is a critical task in modern day systems, and a major area of research and development of new standards (IEEE 1547) [5], [6].

2.1.2 Need for observability and monitoring

One of the biggest problems related to high DER integration is the lack of observability over LV and MV distribution networks. Historically, there was no need to monitor LV feeders in real time, a handful of sensors in MV substations that monitored loads were sufficient as networks were passive. With the introduction of DERs, the limited visibility is a major problem, monitoring the real-time output of intermittent distributed generation, and the active power flows at the distribution level is essential to avoid system failures due to over-voltage, phase imbalance or thermal overload. However, it is not operationally practical nor economically feasible to fit thousands of LV lines with sensors, so DSOs must rely on innovative solutions, like system state estimation (to infer unmeasured voltages and currents) from data provided by advanced metering infrastructures (AMI) and weather forecasts, for example.

A recent event in the UK that highlighted the significant weaknesses of the system caused by DER integration and poor observability was the August 9th, 2019, low frequency event. A single lightning strike led to a cascading failure that caused the largest power disruption in the UK in

over a decade, affecting more than 1.15 million customers. This incident involved the disconnection of two major generators, Hornsea offshore wind farm and Little Barford gas plant, which totalled a loss in generation of 1,378 MW, but more critically, also led to the disconnection of around 1,500 MW of distributed generation (caused by the disconnection of rate of change of frequency (RoCoF) protection). When the low frequency demand response subsequently tried to shed 892 MW of demand to stabilize frequency, only 350MW of net reduction was observed, as 550MW of distributed generation (which was unmonitored) also was disconnected. Subsequent analysis showed that retaining only 150 MW of that distributed generation could have kept frequency within critical thresholds and therefore avoided the further shedding of demand [1].

This event, along with several other incidents, highlights the importance of efficient and scalable visualization methods for LV and MV grids. System operators and planning engineers require readily intelligible data to make informed critical decisions, both for real-time operations, which require decisions within seconds and for infrastructure planning and handling connection requests, for which a clear topological view of the grid is crucial to make well-informed decisions. With the increased operational challenges in LV grids already discussed, enhancing these visualization and observability capabilities is necessary for effective, modern smart-grid management.

2.2 Visualization of Power System Data

2.2.1 Evolution of visualization systems in power grids

Power grid visualization has undergone a significant transformation, from simple analog displays used in the early 20th century, to sophisticated systems that handle massive streams of data in real time. The focus of this research section is to reveal how technological evolution has fundamentally changed how utilities monitor, control and optimize their increasingly complex electrical infrastructure.

The first systems related to visualization started being used in the 1920s, with rudimentary SCADA systems where operators relied on dials, indicator lights and hand switches to monitor and control high-voltage substations. These early analog devices provided visual feedback on system status, and were often overlaid on boards featuring physical diagram drawings of the grid layout. Orders to field personnel were transmitted via engine telegraphs, reflecting the earliest stages of grid control and coordination [7].

The 1960s marked one of the most important transition periods when digital computers began replacing legacy analog systems. Early digital SCADA systems emerged that were built on costly hardware and proprietary networks, with limited scalability. However, some years later, the introduction of remote terminal units (RTUs) around the 1970s significantly improved the scalability and functionality of these systems. The 1980s decade also marked the introduction of the term intelligent electronic devices (IEDs), which laid the foundation for more automated grid operation. It was also over these decades that cathode ray tube (CRT) displays started to be used along computerized systems, which revolutionized the display of information [7].

The 1990s represented another breakthrough period, with the introduction of internet protocols, and graphical user interfaces (GUI). TCP/IP protocols allowed previously isolated SCADA networks to transition into interconnected interoperable platforms, allowing seamless control between different substations and control centers. The widespread GUI adoption also revolutionized how information was presented, enabling a more responsive and operator friendly environment.

2.2.2 Geographical vs topological references

Contemporary grid observability architectures, including SCADA, GIS, ADMS and related applications, rely on two complementary spatial frameworks: geographical, which maps data and grid elements onto their precise locations, and topological, which models the networks connectivity independent of geography. Each of these approaches is more suitable for different applications, as each has its own advantages and disadvantages depending on the system's characteristics and the data displayed. Additionally, there exist hybrid approaches that try to capture advantages of both [8].

Geographical reference: based on geographical information systems (GIS), system elements and their associated data are displayed anchored to their physical location. This approach is useful when the location of the information shown is relevant, and it can help pinpoint the geographical coordinates of a source of information, which is especially useful for example when determining fault locations in transmission lines, or supporting street level maintenance planning, easing the process of dispatching field crews. However, it might struggle to represent information in an organized and structured manner, failing to convey information effectively when a significant zoom is necessary in areas like dense urban networks, where information might be too cluttered otherwise, while the opposite occurs in rural areas, where information might be excessively sparse [8].

Topological reference: focuses on displaying electrical connectivity rather than physical geographical distance. This helps to identify the relationships between different elements of the network, like separating different voltage levels, and how system dynamics in an area of the grid might spread to neighboring grid elements, or how a local power outage could spread to adjacent areas and what switches could be potentially operated to isolate it. Single-line diagrams remain the industry standard for this purpose, thanks to their ability to express bus-branch relationships in a compact format, scaling from substations to interconnections [8].

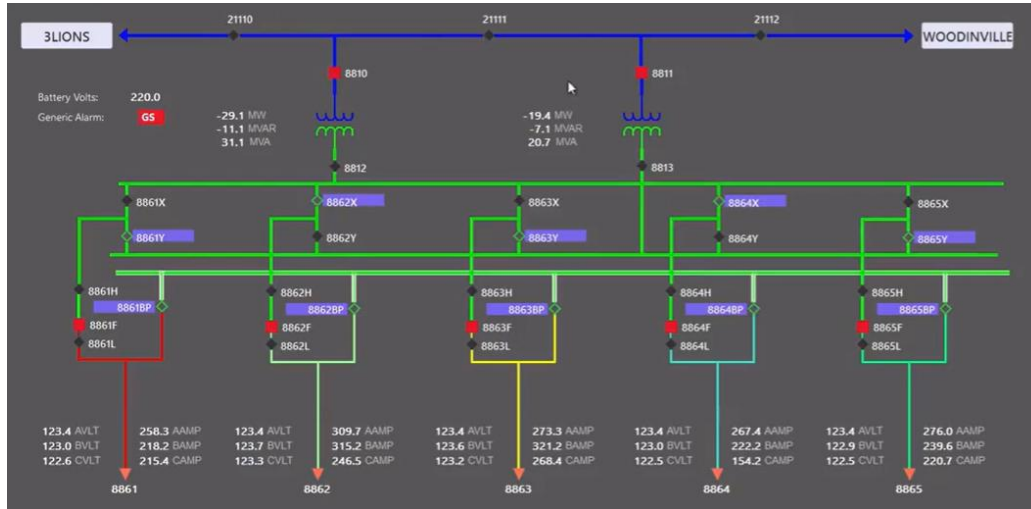


Figure 2-2: Substation's topological single-line diagram from GE's ADMS [9]

Hybrid reference: for increasingly large and complex networks, purely topological references do not scale well, as losing the geographical reference altogether might be impractical when visually processing large areas of the system. For this reason, a combination of geographical and topological references is likely the most useful approach when representing system information of large-scale grids. This is the approach used to represent grid elements in most SCADA systems, which contain diagrams that retain some information about relative geographical positions, and distance proportionality is sometimes preserved. However, some judgement is required about whether to prioritize physical distance or electrical distance, and up to what extent, which makes the automation of generation of these visualizations complicated and non-trivial [8].

2.2.3 Overview of current approaches to display grid data

Sources of information to be represented in power system visualizations can come from a variety of sources including sensor data (voltage, current, temperature, etc.), simulations, like power flow calculations, or external data providers like weather forecasts, between other sources. These sources of data constitute the information backbone from which operators and planners obtain situational awareness of the grid.

Integrating and displaying all the collected data is however a challenging process, valuable representations must ensure that the data is intelligible and easy to process. Currently, a wide range of approaches to display power system information exist, from traditional single line

diagrams with overlaid data like power flow results to significantly more complex visualizations [8].

Line flow visualization

Line flow visualization consists in overlaying the real-time flows and percentage loading of the various transmission and distribution lines. It can be applied to both geographical and topological reference frames. This can however become challenging for large scale systems, and it is more practical to apply this to a reduced number of lines, so that information is easier to interpret and represent. Power flow through lines is usually represented with arrows over them, showing directionality and that can vary in size and/or color to show the relative loading of lines. Animations can further enhance the information presented, providing a clearer view of the direction of power flow, for example [9].

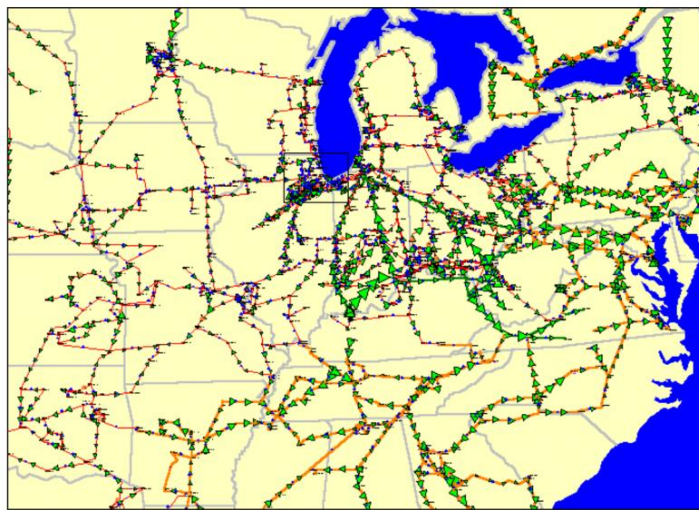


Figure 2-3: Line flow visualization of the US transmission system [9].

Contouring:

Single-line diagrams with numerical displays next to each bus have been used for decades to display system variables. This method has the advantage of representing highly accurate values on precise locations of the grid. However, if it is desired to observe values at a significant number of buses simultaneously, this is not a practical approach as it does not allow to observe large-scale patterns.

A solution to this is to use contouring, which consists in overlaying a colored contour graph (similar to a heat map), that represents a continuous variable like voltage. These variables are commonly not spatially continuous, as they might only exist at buses. It is therefore necessary to create virtual values that span over the entire contour region. The resulting graph contains a color code and a legend to indicate what range of values each color represents. The resulting graph is then overlaid on the corresponding representation of the grid, which can be either geographical (most commonly) or topological [8], [9].

Some typical values commonly represented by contouring traditionally include bus voltage or temperature. However, several other variables can be represented through this method, provided they are continuous. An approach which is popularly used in electricity markets consists in representing locational marginal prices over the grid, as in the example shown in Figure 2.4.

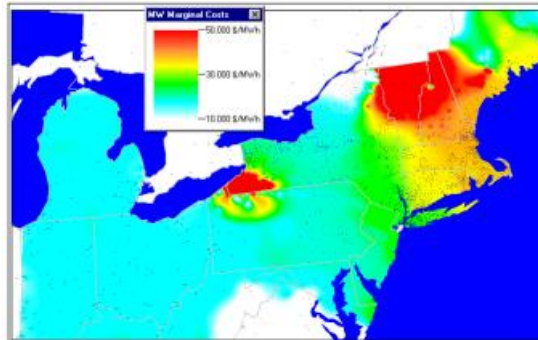


Figure 2-4: Contour of locational marginal prices in New England [9].

Contouring can also be applied to line data, to create similar visualizations to the line flow visualizations discussed before. Contouring however has the advantage of allowing to represent the variation of continuous variables along a single line or a succession of them, which is especially useful along long transmission lines, to represent for example, voltage drop over the length of the line [9].

Another useful visualization enabled by line contouring is power transfer distribution factor, which, for an increase in the power flow between a specific source node and sink in the grid, measures what percentage of that power flow increase would flow through each line in the network [9]. An example is shown in Figure 2.5.

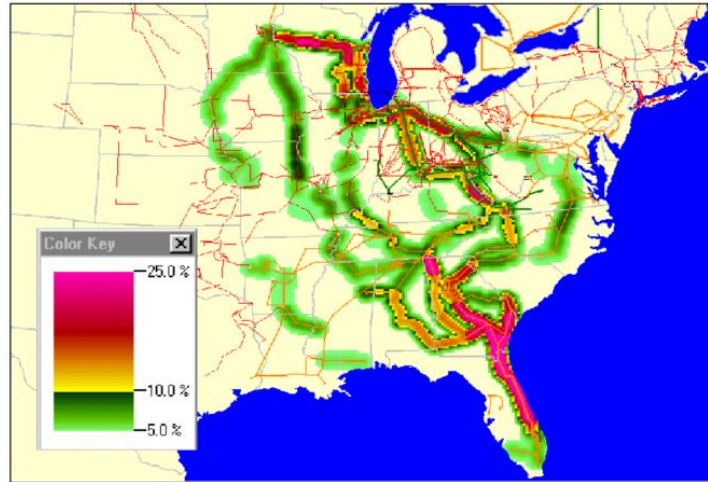


Figure 2-5: Line contouring representation of power transfer distribution factor [9].

Other innovative approaches to visualization

There are many other different approaches for overlaying information over grid representations, and the methods used by different TSOs and DSOs to effectively represent data can vary significantly.

GDV

An additional example is Geographic Data Visualization GDV, a technique introduced by Overbye et al.[10] that allows for representation of a wide variety of variables or system characteristics in a highly customizable way. The technique consists of representing information derived from a power system model using geographical information, in dynamic displays that can be easily customizable. This allows to create visualizations like the one shown in Figure 2.6, in which the rectangles represent generator reactive power reserves, with the size representing the generator's capacity, and the color indicating its reactive power reserves [10].

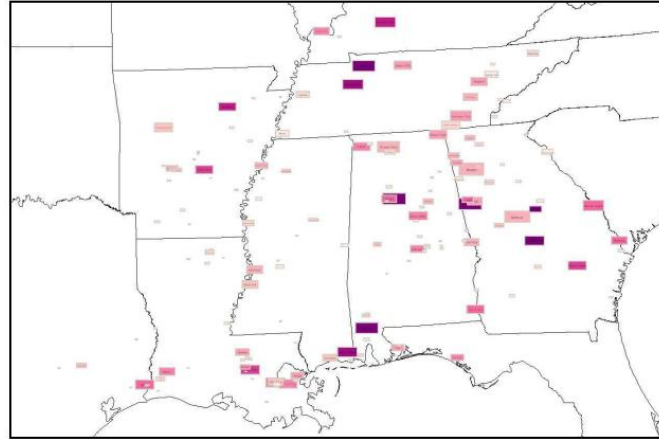


Figure 2-6: GDV displaying power reserves in the US [10].

GreenGrid

GreenGrid is a graph-based transformation algorithm that, using a layout algorithm, creates visualizations that transform geographic views into pseudo-topological layouts in which the line lengths are distorted proportionally to their impedances and elements are moved closer or further depending on either voltage or phase metrics. It is used to identify clusters visually that might indicate, for example, voltage-angle stress, islanded areas, or overloads, which can all be computed in near-real time. This visual geometry approach can be especially useful when observing the behavior of very large systems (the original paper showcases the solution on a 50 000 bus model). Figure 2.7 shows the results of applying the algorithm on a section of the WECC grid, in western USA [11].

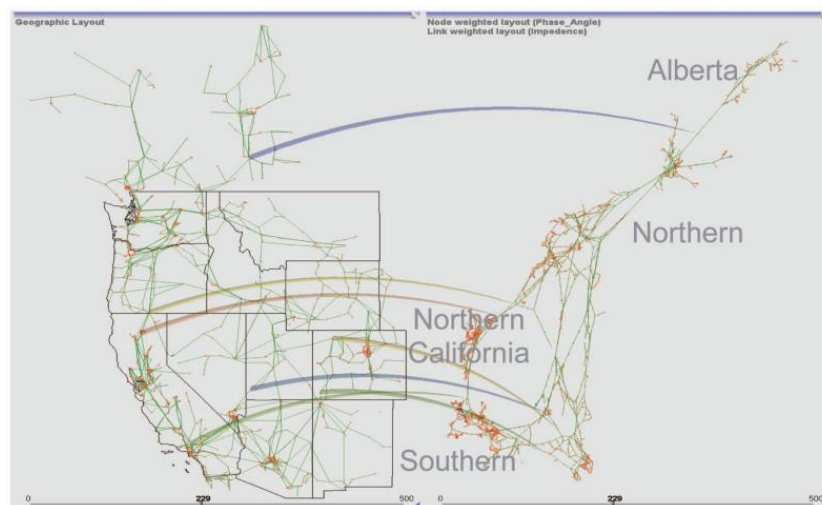


Figure 2-7: GreenGrid's output (right) for a section of the WECC grid (left) [11].

GreenCurve

Another solution, proposed by the same researchers as GreenGrid, is GreenCurve, which rather than creating a graph layout based on the original, completely abandons the graph representation, and uses a fractal, space filling Hilbert curve, that respects electrical distance and in which every node is represented by a single pixel. The color of each pixel is then adjusted based on underlying system variables like voltage, temperature or phase, in a visual 2d grid, similarly to a contour graph. Similarly to GreenGrid, this approach is suitable for observing pattern changes in very large systems, where observing the values of individual buses would not be practical. Figure 2.8 contains the algorithm's output for the same WECC power grid section as used in Figure 2.9, displaying phase angle [12].

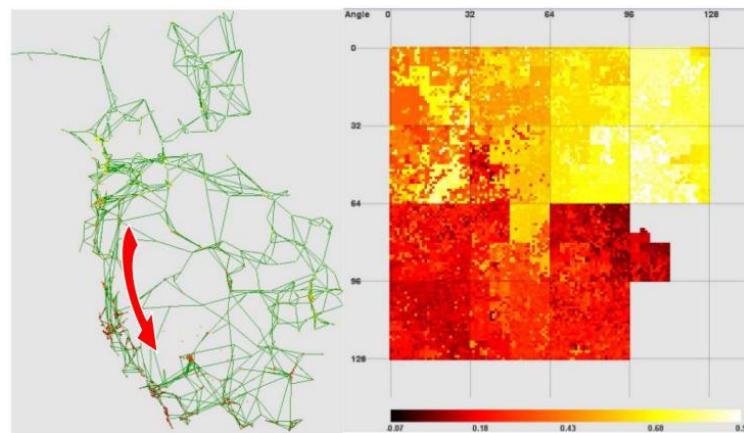


Figure 2-8: GreenCurve output for a section of the WECC grid, displaying phase angle [12].

The authors argue that the two techniques can be applied to obtain synergistic near-real time analysis of the grid, as each will succeed in highlighting different patterns and relationships between variables over the grid, with GreenCurve providing breadth (multivariate coverage), and GreenGrid providing depth (structure-aware focus), making a substantive and innovative contribution to the state of the art in power grid visualization [12].

2.2.4 Innovative approaches

In the 5-year window from 2020 to 2025, AI and machine learning capabilities for the utility sector have advanced rapidly and are now integrated into widespread applications and solutions that system operators rely on. These developments aim to optimize grid operations, improve efficiency, and address the added complexities introduced by the growing penetration of

distributed energy resources (DERs). Deep neural networks, for instance, are increasingly deployed to forecast variable generation with high accuracy, an essential capability as the renewable share in the energy mix, as well as distributed generation keeps increasing [13]. Parallel progress is taking place in the field of visualization, where increasingly sophisticated tools are reshaping real-time decision-making [14].

A standout example in this field is ChatGrid, a commercial application developed by PNNL, which enables the use of natural language queries to produce tailored grid visualizations, that dynamically overlay large amounts of data, including generation, power flow, load, and other network information. By integrating natural language processing (NLP) and on-demand visual synthesis, the platform improves operational efficiency, accelerates situational awareness, and helps operators keep pace with the growing volume and complexity of real time grid data. Figure 2.9 contains a snapshot of the application, and an example visualization created through NLP [15].

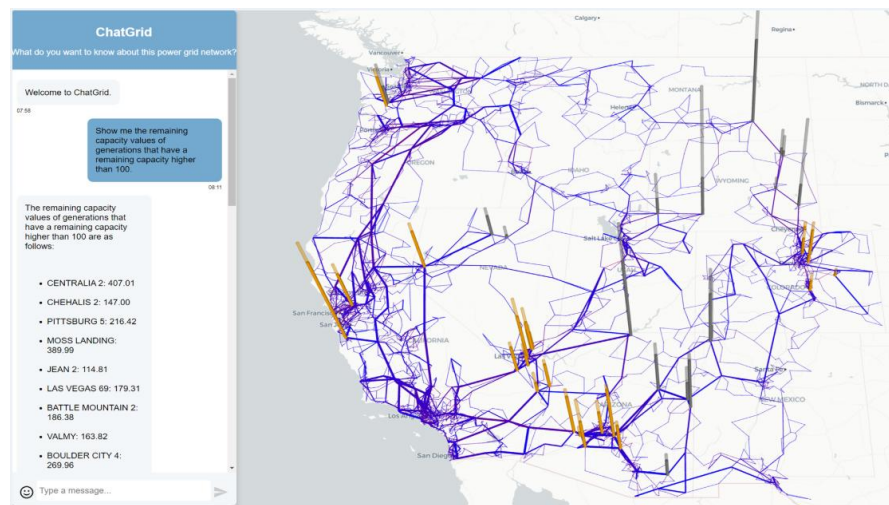


Figure 2-9: ChatGrid dynamic visualization created through NLP [15].

2.3 Modelling LV Networks as Graphs

Due to their inherent network structure, electric networks can be naturally modelled as graphs. In this mathematical abstraction, electrical components and connection points are represented as nodes, while cables, including transmission and distribution lines are represented as edges. This representation provides a powerful mathematical framework for analysing and visualizing

complex electrical networks, thanks to the application of well-studied graph algorithms [16]. Commonly, these power system graphs consist of:

- Nodes: representing buses, substations, transformers, generators, loads, cable joints, and other electrical equipment.
- Edges: representing transmission lines, distribution feeders, cables and other electrical connections between components.

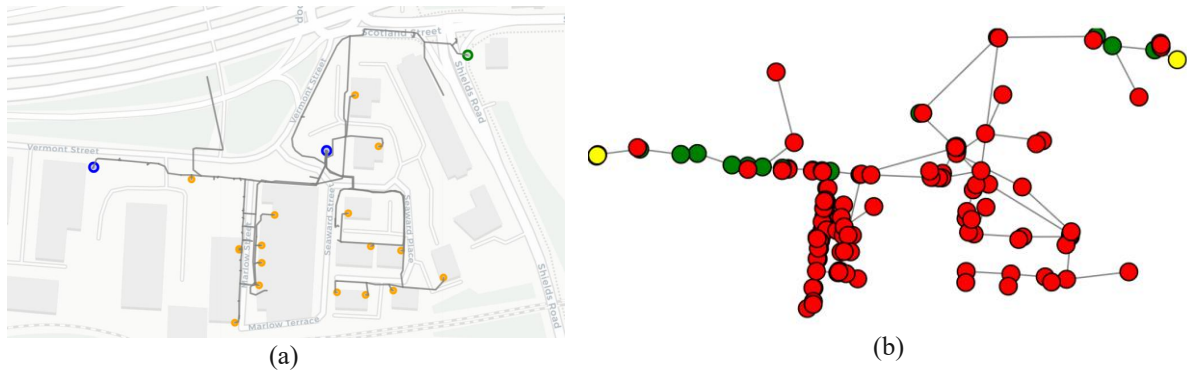


Figure 2-10: (a) GIS view of an LV feeder (b) Corresponding graph representation.

2.3.1 Applications of Graph Theory in Power Systems

The graph-theoretic framework enables numerous applications in power system analysis and operation. Some of these applications include:

Topology Processing: graph algorithms efficiently identify network connectivity, perform efficient and fast adjacent component checks, and potentially network-scale state estimation. Depth-first search and breadth first-search (BFS) algorithms enable the detection of energized sections and trace power flow paths. This has a wide range of applications in power systems like detecting islands and meshed or radial network sections [17].

Power-flow and optimization: graph formulations can enable algorithms that directly operate on the graph structure to solve optimization problems, including optimal power flow, loss minimization or voltage regulation. Othman [Power Flow Solution] (2022) proposed a “Flow-Augmentation PF” replacing the traditional Newton-Raphson iterative method, and achieving 70% faster solves on the IEEE-118 network. Additionally, unsupervised GNNs have been demonstrated to directly approximate OPF solutions four orders of magnitude faster than traditional solvers, thanks to acting on a nodes path and neighbors rather than on the full admittance matrix [13].

State estimation: graph modelling provides state estimation algorithms with two important advantages. First, modifications in the incidence matrix can be calculated on the fly from the adjacency list, so topology changes (like a switch opening/closing) can be reflected instantly without having to rebuild large Jacobian matrixes. Second, topological observability tests are reduced to purely path checks through DFS, much faster than numerical rank calculations for every estimator operation [17].

Network Reconfiguration and service restoration: for strictly radial distribution systems, any switching plan must leave the network cycle-free. From a graph theory approach, this can be approached as an optimization problem, that finds a tree that minimizes losses subject to certain operational constraints. Solving this optimization problem can be fundamental in choosing a switching plan to restore service after faults [18].

Visualization: widely studied graph layout algorithms can be applied directly to graph representations of power networks, creating useful visualizations of the underlying graph topology and characteristics. Some implementations, for example, include the use of force-directed layouts that allow representing large grids in pseudo-geographical positions that avoid overlapping between nodes(substations) and edges (HV lines) [19].

2.3.2 Advantages of Graph-Based Modelling

Modelling power systems as graphs offers several advantages for power system analysis:

Computational efficiency: graph algorithms generally have well-studied computational complexity, enabling scalable solutions for large networks. Many power system problems can be reformulated as classical graph problems with already established efficient solutions [17].

Modelling and simplification: graphs can be used to model complex electrical behavior into topological relationships, making it easier to understand network structure and develop intuitive visualizations, for example. A useful example is how graphs allow encoding directionality in their edges, which can be used to model relationships specific between components which might have a restricted power flow direction, like transformers or HVDC converters, or the intended single-direction of power flow in radial networks [16].

Algorithm reusability: thanks to extensive research in the field of graph theory, a wide of algorithms are available to be adapted to power system applications. From shortest path

algorithms for fault location to graph layout algorithms for visualization, many established techniques can be useful for power network applications [19].

2.3.5 Challenges and Considerations

Although graph modelling provides numerous benefits, several challenges must also be considered and addressed when using graph representations of power systems:

Loss of electrical detail: pure topological representation can potentially overlook important electrical characteristics. For instance, two electrically distant buses might appear adjacent in a graph visualization, potentially misleading operators about electrical adjacency. Using impedance-based weights for edges and considering weights for creating the layout or employing layered hierarchical layouts can help create visualizations that mitigate these issues [16].

Dynamic behavior: static graphs cannot capture time-varying features of power systems. Dynamic graph representations or several graph snapshots are required to capture switching operations, protection actions, and varying load conditions, which could potentially make graph implementation approaches more complex in specific applications.

Scalability vs detail: as networks grow larger, the trade-off between comprehensive representation and visual/computational practicality becomes critical. This motivates network simplification approaches, where intelligent graph reduction preserves essential connectivity while improving visualization clarity [16].

2.4 Geographic Information Systems in the electric power sector

Geographic Information Systems (GIS) are integrated frameworks of software, data and hardware designed to capture, manage and analyse spatial (geographical) information. In simple terms, a GIS links data to specific locations on maps, allowing users to visualize and interpret data with a geographic context [20]. This allows to map features on the earth's surface, from roads and terrain to infrastructure and demographic data, which enables analysis and better decision-making.

In the electric power sector, GIS provides digital mapping and locations of the electric grid infrastructure and components, serving as a foundation for many operational and planning activities. Combining geographic maps and asset databases and analytical tools, GIS allows utilities to keep track of where their assets are located and store specific attributes (e.g. status, age...) for each of them [20].

2.4.1 GIS in electric distribution networks

At the distribution level, GIS has significant impact. Distribution networks are vast and complex, comprised by thousands of miles of feeders and a wide variety of electrical equipment spread across neighborhoods. Utilities have adopted GIS to create digital maps of these distribution grids, mapping every asset in a spatial database.

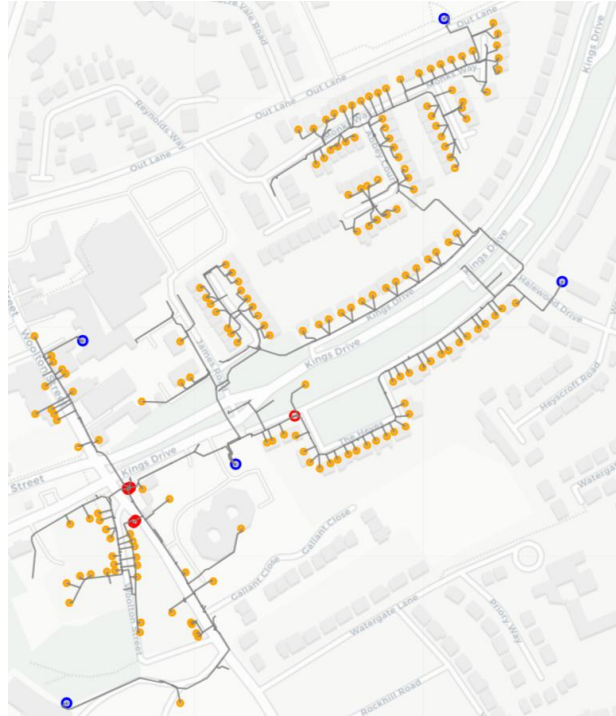


Figure 2-11: GIS Representation of a SPEN LV Feeder.

This geospatial network model is fundamental for daily operations; it allows utilities to visualize the entire electrical network geographically and trace how every customer is connected. By maintaining an up-to-date GIS database of network assets, utilities can easily access information on the location and attributes of equipment, improving situational awareness and asset management.

GIS integration has significantly improved the efficiency and accuracy of power system analysis and operations. These efficiency improvements have translated into significant operational savings over the past 20 years for utilities compared to traditional manual methods. Currently, GIS serves as the digital backbone of advanced grid applications in the power sector, essential for the daily operations for utilities, including ADMS, DER management and real-time outage restoration [21].

One of the most relevant formats used for the exchange of data between GIS and other utility web-based applications is GeoJSON. GeoJSON is an open-standard geospatial data exchange format specified by the IETF in RFC7946. It is built on ordinary JavaScript Object Notation (JSON) and represents spatial objects along with user defined attribute fields. Many commercial and open-source GIS platforms read and write GeoJSON natively [22], [23]

2.4.2 Key applications of GIS in power utilities

GIS supports a wide range of applications for electric utilities, from operational activities to strategic planning. Some of them include:

Asset and network data management: GIS is used by utilities as a comprehensive asset registry and mapping platform. This enables utility staff to locate equipment, monitor asset attributes, and plan maintenance activities with geographical context. GIS also enables digitally updating the network model dynamically, registering new changes instantly, which was not possible with manual physical maps. This dynamic updating also can include attributes linked with the data, which can be sourced from other enterprise systems, like billing, or SCADA [20].

Outage management and emergency response: GIS is essential for handling power outages and improving reliability. Outage management systems (OMS) are built on the connectivity model information provided by GIS. When a fault occurs, the OMS uses the GIS model to determine the most likely fault location. Dispatchers can observe the affected area in real time in a map and quickly deploy field teams to the right location, which significantly improves outage response and reliability indices [24].

System planning and network expansion: planning engineers rely on GIS to make informed decisions about how to expand and reinforce the grid. GIS provides spatial context for infrastructure planning through load growth forecasting. For example, planners can visualize load growth on a map and identify areas that will need capacity upgrades. Common applications also involve using GIS for finding optimal routes for new transmission lines, feeder expansions or substation placement, based on variables like capacity requirements and terrain topology, for example.

2.4.3 GIS as a foundation for grid modernization

GIS has played a key role in the modernization of grids over the past 30 years, thanks to the digitalization of network data. This digitalization has enabled utilities to transition from paper maps to dynamic digital twins of the grid, a digital foundation that supports enhanced analytics, automation, and data-driven decision making. Due to the increased complexity in LV/MV grids caused by DER integration, a solid digital foundation is increasingly necessary, and therefore utilities must clean and refine their GIS datasets to adopt new smart grid systems.

In conclusion, Geographic Information Systems provide an essential platform for electric power utilities. It enables them to manage their networks proactively and more efficiently, supporting the evolution towards a smarter, more resilient power grid [20]. By leveraging GIS, the electric power sector can better meet the challenges of grid modernization, integrating the large volumes of real-time data from current distribution networks, to ensure safe, optimized and reliable power delivery.

2.5 Automatic Schematic Generation of Single Line Diagrams

Power system visualization through diagrams has undergone fundamental transformations from manual paper-based and hand drawn diagrams, to sophisticated digital solutions. Traditional single line diagrams, as defined by IEEE 315, represent electrical circuits using single lines and graphic symbols to show system topology and component relationships. The manual creation process was time consuming and inefficient for large network sections, driving the need for automated solutions.

2.5.1 Evolution of automatic diagram generation

Automatic diagram generation refers to algorithms and systems that produce simplified abstracted diagrams, which can include diagram maps, circuit diagrams or network representations, from raw or structured data. Applications span geography (e.g. transit diagrams, orthogonalized maps), circuitry, and engineering networks (e.g. power systems, water distribution networks), for example [25].

Some early applications of automated diagram generation include the creation of metro-style maps for large transit networks. These applications targeted readability criteria, such as octolinearity (orthogonal and 45° oriented lines), even spacing and minimal crossings, while preserving topology and user recognisability. In early approaches, some methods used for satisfying these constraints included optimization approaches, based on simulated annealing and gradient descent [26].

These optimization approaches are however susceptible to converging to poor local optima, in which the quality of the resulting diagram is strongly dependent on grid spacing, initial positions and the number of iterations [27]. Some more recent approaches include the use of mixed-integer programming formulations, which has the trade-off of significantly higher computational cost and difficulty adding new aesthetic criteria [27].

Schematic generation for electronic circuits has also been explored across multiple design levels, from logic netlist RTL diagrams to circuit layouts. Earlier approaches levered heuristic methods, that relied on placements and routing strategies with value-propagation techniques[28]. More recently, machine learning approaches have become the standard, with solutions incorporating graph neural networks to learn topology and component placement for analog circuits.[29]

The evolution towards power distribution automatic single line diagrams combines aesthetic requirements from geographical network diagrams (like transit systems) and topological orthogonal line routing, as well as diagram component placement requirements from circuit design.

2.5.2 Generation of single line diagrams

The automatic generation of single line diagrams has been researched since the late 1970s, with various approaches developed to address the complexity of modern electrical networks. Several topological and aesthetic constraints and objectives must be achieved in the generation of diagrams for LV/MV grids including minimizing edge crossings, avoiding component overlap and maintaining layout clarity.

Early approaches and foundations

Some early approaches include pioneering work presented by Canales-Ruiz et al.[30] in 1979, proposed an automatic drawing algorithm for one-line diagrams. Their approach formulated the problem as a layered graph ordering algorithm, in which every node represents a bus in the diagram, and the layer for each bus is assigned based on a longest path algorithm. The algorithm then takes an iterative greedy switch approach to swap the edges in each layer of the diagram until it finds a solution with no crossings. This underlying graph is then represented graphically by assigning strictly vertical lines to buses and horizontal lines to connections between buses, as shown in figure 2.12:

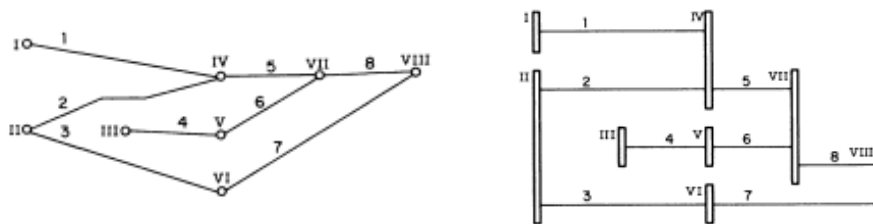


Figure 2-12: (a) Underlying Graph and (b) Equivalent representation proposed by Canales-Ruiz et al.[30].

Other approaches have built on this foundation, treating the diagram representation of Power Systems as a graph layout problem. These approaches can be based in several different layout algorithms that, depending on the constraints and characteristics of the input data, make some more suitable than others. Widely explored algorithms include:

Tree algorithms: some examples like the Reingold Tilford algorithm, are used to represent graphs as rooted tree, useful if the underlying network is completely radial thanks to its simplicity. A rooted tree layout was proposed by Nagendra Nao et al. [31] in 2003, which generates a layered tree layout given a power system radial feeder, and then converts the layout into an orthogonal bus structure representation.

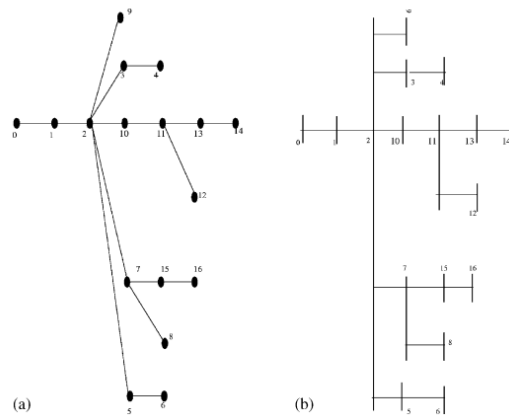


Figure 2-13: (a) Rooted tree layout, and (b) resulting bus structure proposed by Nao et al. [31].

Force directed and spring-based approaches: Useful algorithms that have gained popularity for their ability to handle the layout of graphs with large numbers of nodes. These layout algorithms model the graph as physical system in which nodes repel each other, and edges act as springs pulling nodes together. This method allows to optimize crossings and space out the nodes in the graph over the laid-out space. Birchfield and Overbye [19](2018) used a force directed method to model a layout of a synthetic transmission network with 25000 substation cables with the objective of minimizing overlap and lines crossing over substations. Although these approaches are intuitive and can handle diverse topologies, they do not generate the orthogonal and layered layouts often desired in one-line diagrams; nor do they yield geographically faithful representations.

Integer linear programming and MILP formulations

More sophisticated approaches emerged that treat diagram generation as a constrained optimization problem. Mixed- Integer Linear Programming formulations have been proposed

that guarantee optimal solutions for objectives such as crossing minimization or area minimization [32]. Some constraints that these models can include are:

- Orthogonality requirements.
- Minimum separation distances between components.
- Bus alignment and orientation.
- Hierarchical (layered) structure.

Lin et al. [33] combined a mixed integer linear program to simultaneously assign rows and columns to the connections of urban substation diagrams. This allowed to represent the connections as orthogonal lines with no crossings inside the substation diagrams.

ILP can deliver high quality reproducible diagrams, with tailored constraints for each use case, but their computational complexity grows exponentially with network size, making them impractical for generating diagrams for large scale networks.

Hybrid and Heuristic methods

To address scalability issues, research has been developed on hybrid approaches combining exact algorithms for local optimization with heuristic methods for global layout. For example, Sen et al. [34] proposed an automated generation and incremental update method specifically for distribution networks with rings. Their approach uses a trunk and branch model, where the main feeder is first identified and drawn as a straight line, followed by lateral placement of lines using weight calculations. This method allows creating representations for radial distribution network layouts, in which it is desirable to view the main feeder as a single line from which other connections branch out, and dynamically update those branches as new elements are connected, or older disconnected from the network [34]

Another approach, by Zhou et al. [35] uses a genetic algorithm inside a hierarchical layout engine to position ring cabinets, while penalising crossings. This allows to maintain a hierarchical structure, typical in distribution network diagrams, while minimizing edge crossings and other adjustable flexible optimization criteria.

Machine learning and AI based approaches

Thanks to the most recent advances in machine learning, some applications for single line diagram schematic generation have been developed, which leverage deep learning of layout patterns from existing diagrams [36]. These approaches can capture implicit design rules and aesthetic criteria that are difficult to code into linear programming algorithms. These methods however face significant challenges:

- Limited training data of power system diagrams available
- Topological fidelity constraints might not be satisfied
- Scalability issues

Liu et al. [36] (2025) managed to successfully train a neural network to predict layer assignments and bending points from raw network graphs, producing cleaner layouts.

2.5.3 Commercial landscape and industrial implementations

The most widely extended commercial application featuring automatic single line diagram generation for power systems is DIgSILENT's *PowerFactory*, which integrates an automated diagram generator for visualizing power systems. There is not abundant information available about this tool, however, according to their developer, the tool features: “*Automated drawing of Site and Substation Diagrams*”, “*Diagram Layout Tool for auto-drawing or assisted drawing of full or partial network, feeders, protection devices (CTs, VT, relays), branches, site and substation diagrams as well as auto-expansion of diagram*” and “*User-definable symbols and composite graphics*” [37].

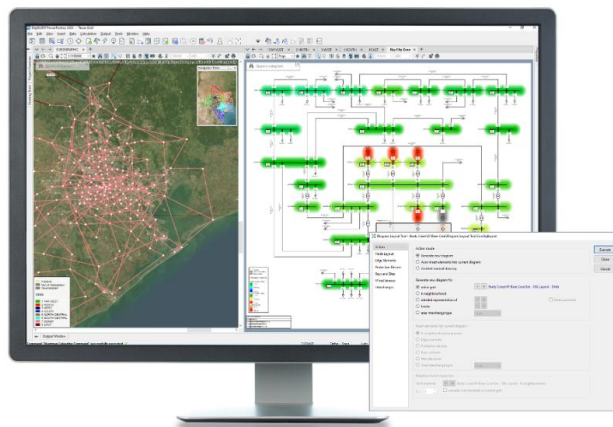


Figure 2-14: DIgSILENT's PowerFactory diagram layout [37].

Although no other commercial widely available tools that automatically generate diagrams from input raw network data are available, some other solutions implement small-scale layout features. For example, ETAP's *Electrical Line Diagram "Auto-Build"* [38] feature allows dynamically creating single line diagrams by simply selecting elements to include in the diagram, while the tool automatically and dynamically creates a diagram layout. PSSE's Diagram Builder offers a similar functionality, aiding the user in creating single line diagrams by automatically arranging the components being added [39].

2.5.4 Current research gap

Although innovative solutions and useful applications have been developed over the past 4 decades, the field of automatic diagram generation is still largely undeveloped. Most contributions target very specific problems that cannot be generalised broadly to provide useful standardized single line diagrams. Some of the current obstacles in this underdevelopment in the field include:

Lack of an end to end open-source tool chain: existing commercial applications like *PowerFactory* embed proprietary routines, and no peer-review study, provides a reproducible step-by-step implementation of a solution that integrates the whole pipeline from raw network data to standard orthogonal single-line diagrams.

Algorithmic bias towards radial or weakly meshed topologies: most academic research with the objective of producing single line diagrams assumes radial networks, while research in meshed networks is not specifically focused on the generation of standardized orthogonal single line diagrams, but rather on producing generic graph drawings, like through force-directed layouts.

Underdeveloped treatment of network simplification: research on network simplification is disconnected from the generation of diagrams. Some approaches implement clustering to create a simplified visualization of the network, but do not focus on developing a diagram on the simplified data. A streamlined pipeline of generating simplified graphs and creating diagrams based on the simplified graphs is not available, although reducing the node count to isolate the key connectivity information in the network before laying it out can potentially provide significant value.

Lack of heuristic implementation on generating crossing free graphs for meshed networks: scalability is essential for real network data, which can have significantly meshed characteristics. However, beyond small instances solvable by integer linear programming, there is a lack of published work on scalable heuristic crossing reduction algorithms that can render large, meshed distribution networks without visual clutter.

2.6 Summary

This chapter has established the context for this project, linking the project's motivations to the current challenges in LV grids, that, due to the increasing penetration of distributed energy resources, calls for improved observability and visualization; and providing a state of the art in current visualization technologies, their evolution, and more specifically, automatic single line diagram generation; as well as explaining the role of GIS, as a digital backbone for electricity utilities worldwide.

Representing LV networks as graphs is another fundamental foundation of this project, and a wider base for graph theory concepts relevant to the project will be explored in the next chapter, before outlining the approach followed for the automatic generation of single line diagrams.

3 Foundations of Graph Theory

This chapter lays out the mathematical and algorithm groundwork that the methodology in Chapter 4 will refer to. It covers several graph theory concepts, from basic definitions to graph layout algorithms. These concepts are defined beforehand to avoid repetition in the explanation of the approach.

3.1 Fundamentals and Core Concepts

Note: all the explanations in this section have been based on the definitions and content of the book Introduction to Graph Theory (fourth edition) by ROBIN J. WILSON [40] and the Dictionary of Algorithms and Data Structures [online] by Paul E. Black [41].

3.1.1 Basic concepts

Definition 3.1 (graph): A graph is formally defined as a pair $G = (V, E)$ where:

- V is a non-empty finite set of vertices (will be referred to as nodes throughout this paper)
- E is a finite set of pairs of V , called edges: $E \subseteq \{\{u, v\} \mid u, v \in V\}$

Figure 3.1 represents a simple 3-node graph, with three edges, connecting all nodes with each other.

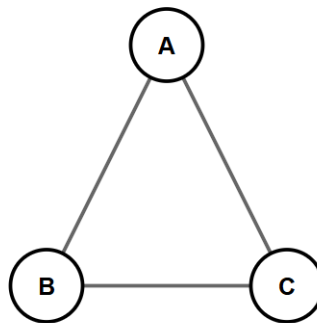


Figure 3-1: Simple 3-node graph.

Depending on whether each edge has an orientation (meaning it is an ordered or unordered pair of edges), a graph can be directed or undirected.

Definition 3.2 (undirected graph): An undirected graph is a graph whose edges have no orientation. Each edge is represented by an unordered pair of vertices, with each edge $\{u, v\}$ connecting vertices u and v bidirectionally, meaning: $\{u, v\} = \{v, u\}$. Figure 3.1 represents an undirected graph.

Definition 3.3 (directed graph): A directed graph, on the contrary, is a graph whose edges have direction. Each edge is represented by an ordered pair of vertices (u, v) , meaning there is a connection from u to v and $(u, v) \neq (v, u)$.

An example of a directed graph with 3 nodes and 3 edges can be observed in Figure 3.2:

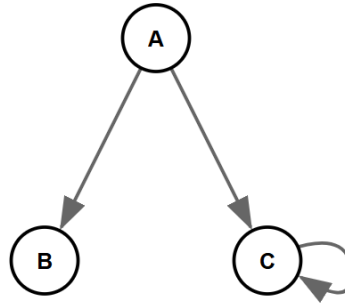


Figure 3-2: Directed graph with a looped edge.

In general graph theory, an edge may connect a node to itself (illustrated by node C in figure 3.2). However, simple graphs, which are the subject of study for this project, explicitly exclude looped edges and multiple edges (more than one edge connecting the same pair of nodes). These restrictions are imposed to ensure that each edge represents a unique non-redundant connection between a certain pair of nodes.

In most applications and algorithms, time complexity scales linearly with the number of nodes and edges, so a common approach is to isolate the section of the network relevant to the task in hand. This isolated section is referred to as *subgraph*

Definition 3.4 (subgraph): For any given graph $G = (V, E)$, a graph $G' = (V', E')$ is called a subgraph of G , written $G' \subseteq G$, when:

- $V' \subseteq V$, and
- $E' \subseteq E \cap \{\{u, v\} \mid u, v \in V'\}$

Meaning that a subgraph G' contains a subset of G 's nodes and edges. Additionally, G' is denominated *induced subgraph* if $E' = \{\{u, v\} \in E \mid u, v \in V'\}$: i.e. all edges in G between nodes V' are preserved in the subgraph. If the subgraph contains all nodes in G , $V = V'$ but omits some edges, G' is denominated a *spanning subgraph* of G .

3.1.2 Graph representations

While visualizing a graph as a diagram of points (nodes) connected by lines (edges) is often the most intuitive form of representation, such a layout is not the most efficient or rigorous for computational purposes. For mathematical analysis and algorithmic manipulation, graphs are represented and stored using formal data structures, such as an edge list, an adjacency matrix or an incidence matrix.

An edge list, or an adjacency list, is a data structure that contains a collection of all the edges in a graph, with each being recorded as a pair of the nodes that it connects. This format is compact and most suitable for sparse graphs, where the number of edges is significantly less than the squared number of nodes. Edge lists are commonly stored in unordered structures, like dictionaries. Figure 3.3 shows the edge list for a simple graph

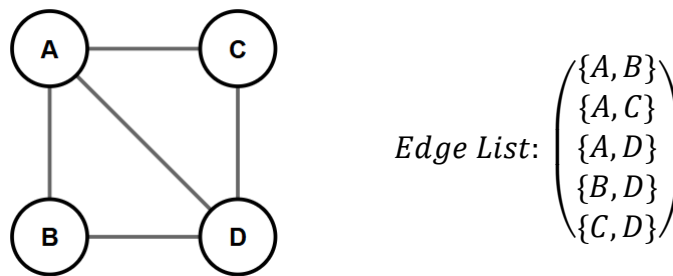


Figure 3-3: Edge list.

An adjacency matrix is a two-dimensional array in which the rows and columns correspond to nodes. The entry at position (i, j) indicates the presence of an edge between nodes i and j . This representation allows for time-efficient edge existence queries but requires $O(n^2)$ space; making it suitable for dense graphs, with a significant number of edges compared to the squared number of nodes.

In an incidence matrix, the rows correspond to nodes and the columns to edges. The entries in each column indicate which nodes are incident to each edge. For undirected graphs, entries are typically binary (0 or 1), while directed graphs may use -1 or 1 to represent directionality. This representation is beneficial when analyzing relationships between edges and vertices, particularly in linear algebraic formulations of graphs.

Figure 3.4 shows an example of a graph and its adjacency A and incidence M matrices representations:

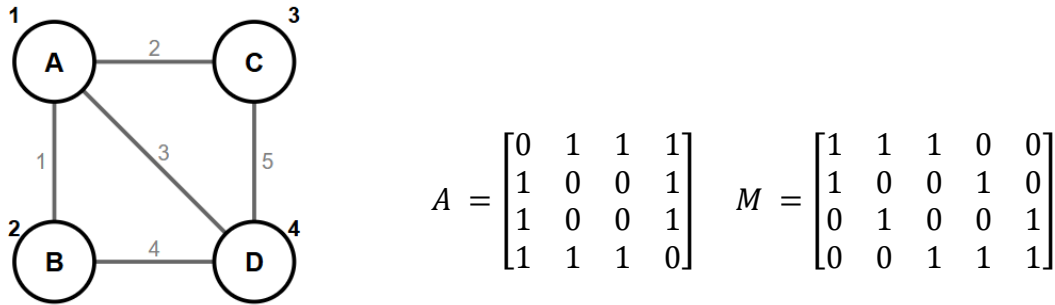


Figure 3-4: Adjacency (left) and incidence (right) matrices.

3.1.3 Graph properties

Definition 3.4 (degree): The *degree* of a node is the number of edges connected to it, which is equivalent to the number of its neighboring nodes. By extension, a *neighbor* of a node is any other node directly connected to it by an edge.

In directed graphs, neighbors can be broken down in two categories, predecessors and successors, and degree into in-degree and out-degree.

Definition 3.5 (predecessor, successor): in a directed graph, a predecessor of a node v is any node u such that there exists a directed edge from u to v , denoted as (u, v) . On the other hand, a successor of a node u is any node v such that there exists a directed edge from u to v , denoted as (u, v) . Additionally, in contrast to total degree, in-degree is the number of predecessors of a node, and out-degree is the number of successors.

In Figure 3.2, node A is the only predecessor of both B and C, and B and C are successors of A.

Definition 3.7 (path): A path P from node u to node v is a finite sequence of nodes (v_0, v_1, \dots, v_k) such that $v_0 = u$, $v_k = v$, and for every $i \in \{0, \dots, k-1\}$, the pair $(v_i, v_{i+1}) \in E$, meaning there exists an edge between each consecutive pair of nodes in the sequence. The length of the path is the number of nodes it traverses, k .

In undirected graphs, as edges do not have orientation, the direction of traversal along the path is not constrained. However, in directed graphs, the path must follow the direction of the edges.

Therefore, for a sequence (v_0, v_1, \dots, v_k) to be a valid directed path, each edge (v_i, v_{i+1}) must be a directed edge, pointing from v_i to v_{i+1} .

In figure 3.5, a highlighted directed path of length = 2 between nodes C and B can be observed:

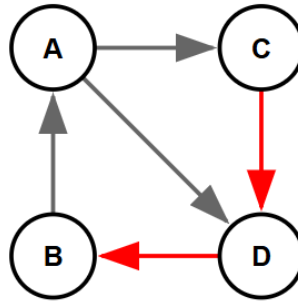


Figure 3-5: Directed path between nodes C and B.

A graph is said to be connected if, for every pair of nodes u and v in the graph, there exists a path from u to v , meaning that there is no isolated node and it is possible to reach every single node through a sequence of edges. For undirected graphs, connectivity simply requires the existence of edges that create a path; however, for directed graphs a distinction is made:

- A graph is said to be strongly connected if for every pair of nodes u and v , there exists a directed path from u to v and from v to u .
- If a directed path does not exist for every pair of nodes in the graph, the graph is said to be weakly connected if the equivalent undirected graph is connected.

Figure 3.6 shows simple examples of unconnected, weakly connected and strongly connected graphs (respectively)

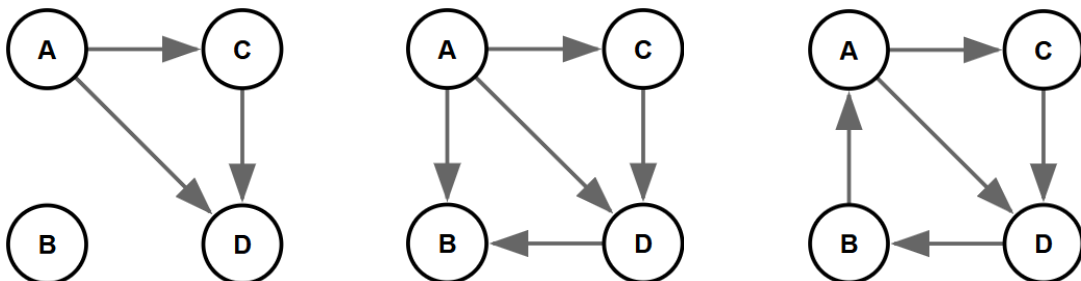


Figure 3-6: Unconnected, weakly connected and strongly connected graph (respectively).

Definition 3.8 (Cycle): A cycle is a special type of path where the starting and ending nodes are the same, and all intermediate nodes are different. Formally, a cycle is a path (v_0, v_1, \dots, v_k) where $v_0 = v_k$ and $v_i \neq v_j$ for all $i \neq j$, $0 < i, j < k$. Cycles can occur in both directed and undirected graphs, with the requirement that edge directions are respected in directed graphs.

The strongly connected graph shown in Figure 3.6 shows a graph containing a cycle, as there exists a path to and from B, that goes through A and D.

3.1.4 Graph traversal algorithms

Graph traversal algorithms are the foundation for many graph operations, that include for example, connectivity testing, cycle detection or path computation. Two graph traversal algorithms are the most widely used, breadth first search (BFS) and depth first search (DFS).

Both of these algorithms have a time complexity of $O(V+E)$, as they are guaranteed to visit each node exactly once, and each edge at most twice (examined once per endpoint)

Breadth first search (BFS):

BFS explores the nodes of a graph in increasing order of their distance (in number of nodes) from a given starting node, meaning that it visits every node at a distance k , before visiting any nodes at a distance $k+1$.

Algorithm 3.1 shows how the algorithm performs this process, using a queue into which the unvisited neighbors of each node being visited are added. This example constructs a *parent* array, that after execution contains the identifier for every node in the graph, explored in the BFS order; however, any other operation that involves visiting all the nodes in this order can be performed.

Algorithm 3.1: Breadth-First Search (BFS)

Input: Graph $G = (V, E)$, start node $s \in V$
Output: parent array encoding the BFS tree

```

1 foreach  $v \in V$  do
2    $\text{visited}[v] \leftarrow \text{false};$ 
3    $\text{parent}[v] \leftarrow \text{nil};$ 
4  $\text{visited}[s] \leftarrow \text{true};$ 
5  $\text{enqueue}(Q, s);$ 
6 while  $Q \neq \emptyset$  do
7    $u \leftarrow \text{dequeue}(Q);$ 
8   foreach  $v \in \text{Adj}(u)$  do
9     if  $\neg \text{visited}[v]$  then
10       $\text{visited}[v] \leftarrow \text{true};$ 
11       $\text{parent}[v] \leftarrow u;$ 
12       $\text{enqueue}(Q, v);$ 
13 return  $\text{parent};$ 

```

Figure 3.7 shows how a BFS algorithm visits a 7-node graph, with the already visited nodes highlighted in green and starting with node A.

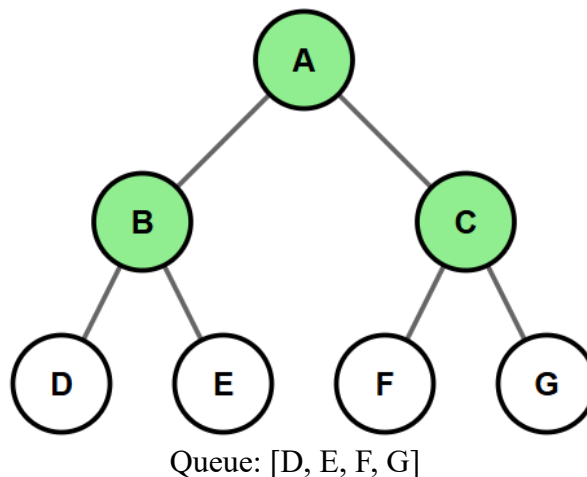


Figure 3-7: BFS algorithm.

The algorithm starts by visiting node A, after which it is marked as visited and its neighbors B and C get added to the queue. Then, node B, the first in the queue is visited, adding its unvisited neighbors (D and E) to the queue while being marked as visited. The next node in the queue is C, so its unvisited neighbors (F and G) get added to the queue, and node C is marked as visited.

The next node that would be visited by the algorithm would be D, the first in the queue. Marking the nodes as visited when adding them to the queue is crucial, as not doing so is a common pitfall that can result in the same node being added to the queue several times.

One of the most relevant applications of BFS algorithms is, for example, finding the shortest path between two nodes, as the first discovery through BFS of a valid path from nodes u to v yields the shortest possible path between the two.

Depth-first search (DFS):

DFS explores the nodes of a graph as far as possible along each branch before backtracking. It prioritizes deeper unexplored paths first before returning to explore other branches closer to the root node.

The algorithm works in a similar way to BFS, but using a stack rather than a queue, so that the last elements added are popped first in later iterations. This is shown in Algorithm 3.2, which creates an array containing all the nodes in graph G , visited in DFS order.

Algorithm 3.2: Depth-First Search (DFS)

Input: Graph $G = (V, E)$, start node $s \in V$
Output: parent array encoding the DFS tree

```

1 foreach  $v \in V$  do
2    $visited[v] \leftarrow \text{false};$ 
3    $parent[v] \leftarrow nil;$ 
4  $push(S, s);$ 
5 while  $S \neq \emptyset$  do
6    $u \leftarrow pop(S);$ 
7   if  $\neg visited[u]$  then
8      $visited[u] \leftarrow \text{true};$ 
9     foreach  $v \in Adj(u)$  do
10      if  $\neg visited[v]$  then
11         $parent[v] \leftarrow u;$ 
12         $push(S, v);$ 
13 return parent;

```

Figure 3.8 shows how a DFS algorithm visits a 7-node graph, with the already visited nodes highlighted in green and starting with node A.

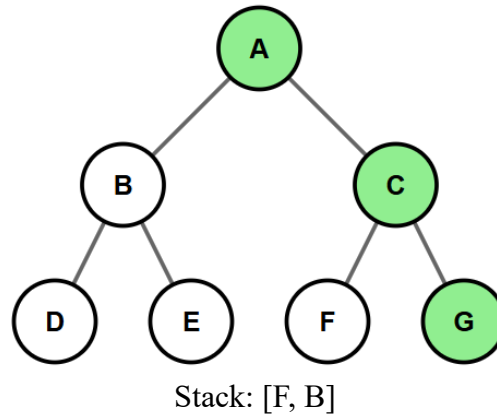


Figure 3-8: DFS algorithm.

The algorithm starts by visiting node A, after which it is marked as visited and its neighbors B and C get added to the stack. Then, node C, the first in the stack is visited, adding its unvisited neighbors (F and G) to the stack and being marked as visited. Then, the algorithm visits the last node added to the stack G, and marks it as visited. The algorithm would then proceed with visiting node F, as it was added the last to the stack before G.

A common application of depth-first search algorithms is cycle detection. DFS is well suited for this task because it explores individual paths deeply, making it more efficient at identifying back-edges, compared to breadth-first search.

Definition 3.9 (back edge): a back edge is an edge connecting a node to one of its predecessors.

Formally: $e = \{u, v\} \in E$ is a back edge $\Leftrightarrow (u < \text{pred}(v)) \vee (v < \text{pred}(u))$.

Meaning that e is a back edge if u is a predecessor of v and v is also a predecessor of u .

The existence of back-edges indicates the existence of a cycle.

3.1.5 Directed acyclic graphs

A directed acyclic graph (DAG) is a directed graph that contains no cycles, meaning that there is no sequence of directed edges that starts and ends at the same node, while respecting edge direction. If a graph G is a DAG, it means that there exists a topological ordering for that graph, and that DFS on G produces no back edges.

Definition 3.10 (Topological order): a topological order of a DAG $G = (V, E)$ is a linear ordering of its nodes that satisfies that for every directed edge $(u, v) \in E$, u appears before v in the ordering.

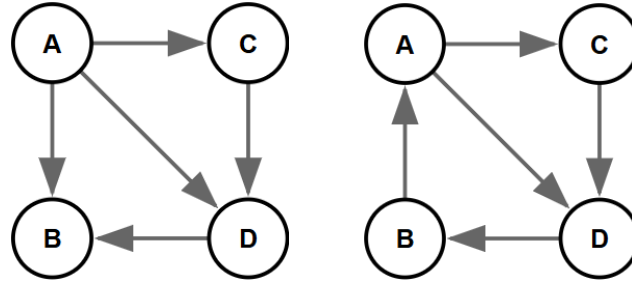


Figure 3-9: DAG and cycle-containing directed graph (respectively).

Figure 3.9 shows two similar graphs, however the left graph is a DAG, while the graph on the right is not because it contains a cycle. Edge (A,B) is swapped between the two examples, so while on the left graph, there is no path connecting a node in the graph to itself, in the right graph there is. These are the same example graphs used in the connectedness section.

Graphs that contain cycles and therefore are not DAGs, can potentially be converted to DAGs by either one of these two approaches:

- Swapping the direction of one or several edges: although this is not a viable solution in all applications, sometimes, if the edge direction is not constrained by a property that must not be reinterpreted, some edges may be reversed to eliminate cycles. This is a viable solution for graphs whose underlying directionality is not explicitly constrained, which for example can allow the use of hierarchical layout algorithms.
- Removing a minimal set of edges so that the remaining graph is acyclic. Again, the resulting graph will not retain all the properties of the original one, but this might be a necessary step in some scenarios. That minimal set of edges whose removal makes the graph acyclic is called feedback arc set (FAS).

Finding a minimum FAS is a NP-complete problem. Therefore, exact solutions are generally unfeasible, so suboptimal solutions must be found through heuristic approaches, which can include greedy algorithms (for example, removing edges that participate in the most cycles), or removing all back-edges found when conducting DFS.

In figure 3.10, it can be observed that the FAS of the non-acyclical graph observed before would be edge (B,A) (although edge (D,B) would also be a valid solution), as removing it would cause the cycle to be removed, and the remaining graph to be a DAG.

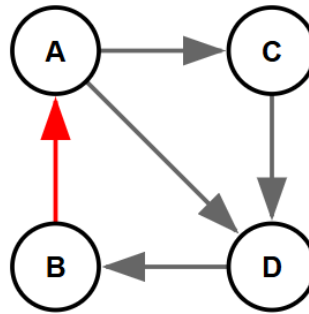


Figure 3-10: FAS of non-acyclical graph.

3.1.6 Planarity

A graph is planar if it can be laid out in a 2D plane such that no edges intersect each other except at their endpoints. In other words, if a graph is planar it can be drawn without any of its edges crossing.

To check for planarity, there are two main alternatives:

- Kuratowski's theorem, which establishes that a finite graph is planar if and only if it does not contain a subgraph that is a subdivision of K_5 or $K_{3,3}$ (shown in Figure 3.11).

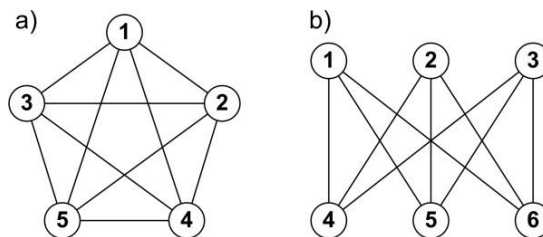


Figure 3-11: Evaluation of topological planarity and reliability for interference reduction in radio sensor networks.

- Euler's formula: establishes that a graph $G = (V, E)$ is planar if and only if it satisfies the following formula:

$$V - E + f = 2$$

Where:

V = number of nodes in the graph.

E = number of edges in the graph

F = number of faces. A face is a region enclosed by the edges in the graph, which includes the outer region.

There are several algorithms that can test for planarity, achieving linear time complexity ($O(n)$), including for example the Hopcroft-Tarjan method (Path Addition Method) or the Boyer-Myrvold method (Edge Addition Method), which is one of the most widely used algorithms currently thanks to its simplicity.

However, even though checking for planarity is simple, finding a layout for a planar graph with the minimal number of crossings is not, and there is not a definitive solution that works on every case. This is actually an NP-hard problem that is generally tried to solve through heuristics

3.2 Principles of Graph Layout Algorithms

Graph layout algorithms have the objective of assigning 2D (or 3D) positions to each node and routing edges so that the drawing conveys structure clearly. The wide variety of graphs calls for different layout strategies, the goal of any layout algorithm is to make patterns immediately visible, failing to do so indicates poor performance of the layout algorithm generally.

Common goals of different layout algorithms are:

- Reduce visual clutter, (minimizing edge crossings, edge bends and node overlap).
- Emphasize structure, (hierarchy, tree shape, flow direction, symmetry).
- Aesthetic constraints (even spacing, alignment, balance).
- Represent data attributes (colors, thickness of edges to represent weight, hierarchy layers...)

There is a vast amount of different layout algorithms, each serving their own specific purpose and best suited for a certain type of graph in particular, including for example: hierarchical, tree, force-directed, circular, matrix-based or geographic. However, this chapter will be focused on two approaches in particular: Reingold-Tilford, a simple algorithm suited to represent radial trees, and the Sugiyama framework, used to represent hierarchical graphs.

3.2.1 Reingold-Tilford algorithm

The Reingold-Tilford algorithm produces tidy drawings of rooted trees, with even spacing, non-overlapping and aesthetically centered predecessors above their successors. This algorithm is widely extended for simple radial graph visualizations. The algorithm was formulated in 1981 in the paper called *Tidier Drawings of Trees*, by Edward M. Reingold and John S. Tilford [42]. It offers a computationally simple algorithm, which runs in linear time to the number of nodes and produces aesthetically pleasing results.

The algorithm only works in acyclic rooted trees, in which every node only has one predecessor. Following the terminology used in the original paper, predecessors will be referred to as *parents*, successors as *children* and successors of the same node as *siblings*.

The Reingold-Tilford algorithm aims to satisfy [42]:

- No node overlaps

- Children are always drawn on a y level below their parent
- Parent centered over its children's span
- Subtrees separated uniformly
- Symmetry preserved, equivalent subtrees are drawn identically

To obtain the desired layout, the algorithm assigns y coordinates to each node based on depth (distance from the root node), and x coordinates based on two traversals of the tree, to guarantee no overlaps and ensure every node is centered above its children. The process that this is done through is explained below:

First, a rooted tree is taken, with each node having one parent and n number of children. The nodes in the tree can be processed in any order, as long as all the children of each node are processed before their parent. Figure 3.12 shows an example graph on which the algorithm is going to be demonstrated. For the example, the nodes will be processed in the alphabetical order shown; however, a different order would also be valid as long as every sibling was processed before their parent.

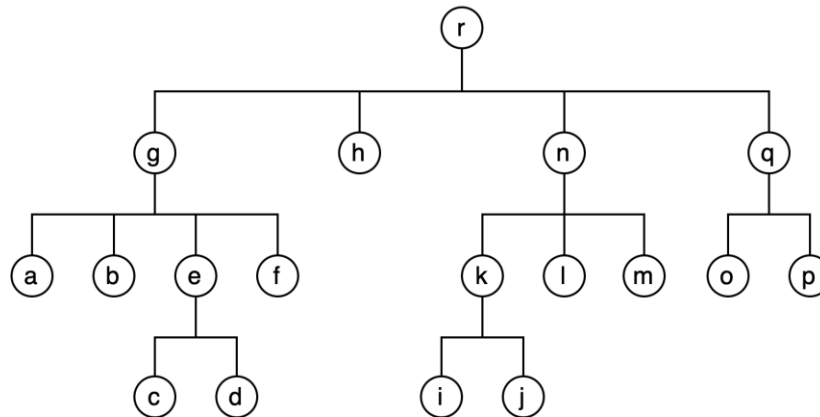


Figure 3-12: Radial rooted tree [43].

First, through an initial traversal of the tree, initial x values are assigned based solely on the number of children of each. The first node of the children is assigned $x=0$ if it has no children itself, or the average position of its two children if it does (0.5 if it has two, 1 if it has 3, for example). The subsequent siblings are assigned incremental x values, with a displacement of 1 between each other, for example, if the first sibling is assigned an $x = 0.5$, the second would be assigned an $x = 1.5$, and subsequently.

For nodes that do have children but are not the first sibling to be processed, their initial assigned x value would not match the middle point among its children, so another variable, called mod , is calculated. The value of mod for any node is later used to later displace the x positions of its children, ensuring every node is centered above its children. This value is calculated as:

$$mod = X_{parent} - X_{(avg\ children)}$$

Figure 3.13 shows the resulting x and mod variables assigned to each node in the example graph. Nodes highlighted in red are nodes that require a displacement of their children by mod

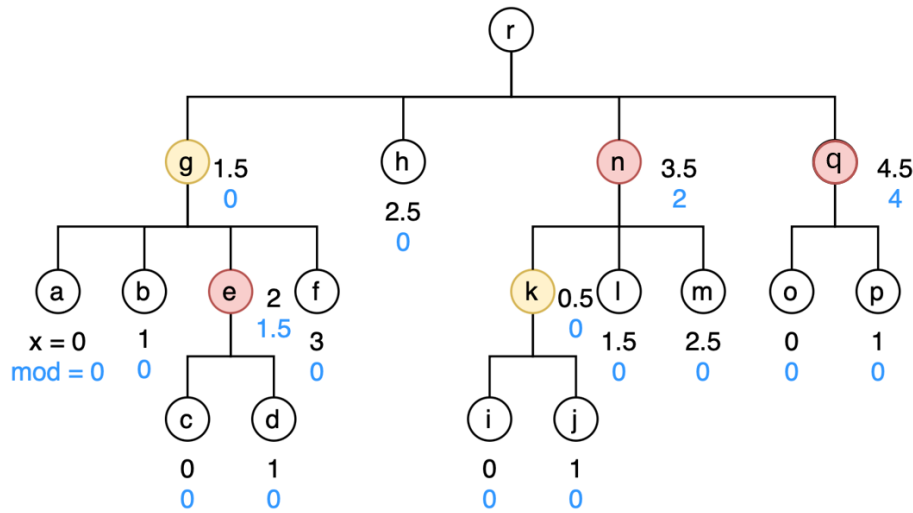


Figure 3-13: Radial rooted tree with mod and x assigned [43].

This node placement, however, does not ensure that different subtrees are prevented from intersecting with each other. For example, the positions of nodes **f** and **k** would cause an intersection between their edges, as **f** would be placed in $x = 3$ and **k** would be placed in $x = 2.5$.

To solve this, the algorithm also performs an overlap detection during its first pass using, what is referred to as in the paper, contour detection. This consists in, for every level and every subtree comparing the leftmost descendant of the right subtree, and the rightmost descendant of the left subtree. If the x position (initial $x + mod$) of the leftmost descendant of the right subtree is not higher than the x position of the rightmost descendant of the left subtree, a shift value is calculated to displace the whole right subtree, so that the overlap is solved. The value of shift is calculated as:

$$\text{shift} = X_{\text{rightmost child (left subtree)}} - X_{\text{leftmost child (right subtree)}} + 1.$$

The X value is calculated considering the mod values of each node's parent, and considering any shift values already calculated for the left subtree. To maintain the constant relative spacing between the nodes at every layer, when shifting a node to avoid collisions, another shift must be applied to siblings of that node. The calculations for all shifts required to avoid the collision between k and f, and maintain constant spacing between siblings are:

$$\text{shift}1_n = \text{shift}_k = 3 - 2.5 + 1 = 1.5$$

$$\text{shift}1_h = \text{shift}1_n * \text{relative position of } h \text{ to } n = 1.5 * 1/2 = 0.75$$

$$\text{shift}1_q = \text{shift}1_n * \text{relative position of } q \text{ to } n = 1.5 * 3/2 = 2.25$$

Another overlap can be observed when comparing the subtree of node q and the subtree of node n. The edges connecting nodes o and m would intersect before considering the previous shift as node o's x position would be 4 and m's x position would be 4.5. After the shift to fix the overlap between g and n subtrees, o and m would not overlap as their positions would be 6.25 and 6, respectively. This, however, violates the constraint of evenly spacing as siblings are normally spaced by 1, but o and m would only be spaced by 0.25, so this situation is solved similarly to an overlap, but with additional consideration, as shifting one of the subtrees would cause a shifting in the other one, to preserve the even spacing. Therefore, to calculate this relative shift necessary to ensure the spacing is at least 1 between nodes, the shift value must satisfy the following equation:

$$(x_o + \text{shift}2_q) - (x_m + \text{shift}2_q * \text{relative position of } n \text{ to } q) \geq 1$$

$$(6.25 + \text{shift}2_q) - (6 + \text{shift}2_q * 2/3) \geq 1$$

$$\text{shift}2_q \geq 2.25$$

Which would lead to a necessary shift in g and n subtrees equal to:

$$\text{shift}2_h = \text{shift}2_q * \text{relative position of } h \text{ to } q = 2.25 * 1/3 = 0.75$$

$$\text{shift}2_q = \text{shift}2_q * \text{relative position of } n \text{ to } q = 2.25 * 2/3 = 1.5$$

No more overlaps are present in the example graph, so after the calculation of these shifts, the first pass would be completed. Therefore, applying the sum of these shifts to their corresponding subtrees results in a layout that has no overlaps, that ensures that the nodes at each layer are each spaced with each other by at least 1, and that siblings are evenly distributed. The final results of all variables calculated in the first pass can be observed in Figure 3.14:

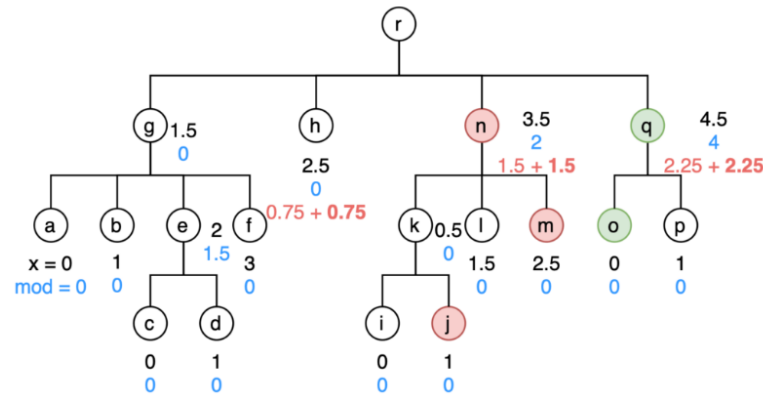


Figure 3-14: Radial rooted tree with mod, x and shift assigned [43].

After the value of initial x, mod and shift for each node are calculated, a second traverse of the nodes is performed to calculate each node's x position based on these three variables. The resulting x position of some nodes could potentially be negative, as the mod variable can take negative values. If this is the case, the lowest x value would be recorded, and after completing the second pass, a third pass would be performed, to subtract that lowest negative value (or add its absolute value) from the x position of every node in the graph. The resulting x positions for this example can be observed in Figure 3.15:

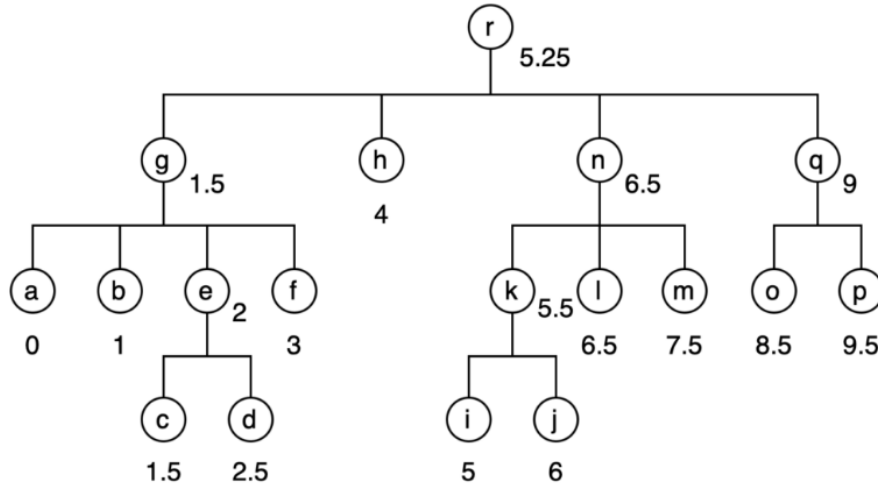


Figure 3-15: Radial rooted tree with the resulting x positions [43].

3.2.2 Hierarchical layouts - The Sugiyama framework

A hierarchical layout is a particular graph layout in which the nodes are placed at discrete y -positions, called *layers*. Laying out graphs in a layered format is essential for visualizing flow-based graphs, so that nodes are organized in discrete horizontal layers and edges generally point top to bottom. This allows viewers to easily understand the flow structure of the graph, trace paths quickly and perceiving source and sink nodes at a glance. These layered layouts are commonly used in applications like control flow graphs in compilers, dataflow diagrams, organizational charts, or circuit diagrams.

Layered graphs and level planarity

Definition 3.10 (Layered graph): a layered graph is a pair (G, ℓ) containing a graph $G = (V, E)$ along with a set of layers $\ell: V \rightarrow \{1, \dots, k\}$, that fixes the vertical position (layer) of each node. A layout that assigns the x position of each node inside each layer is a hierarchical or layered layout.

Definition 3.10 (Level planarity): a layered graph is said to be level planar if there exists a graph layout $L = (G, \ell)$, that for the given set of layers assigning y-positions, the layout contains no crossings between the edges of G .

Detecting level planarity is a complex problem, and most approaches generally involve trying detecting planarity in the generated layouts, after having tried to minimize crossings, not checking planarity before calculating the layout [44]. Figure 3.16 shows a non-level-planar graph, as for the given layer assignment $\{A: 1, B: 2, C: 2, D: 3, E: 3\}$, there exists no layout that can avoid all crossings between layers 2 and 3. Generating a planar layout for this graph would require a different layer assignment.

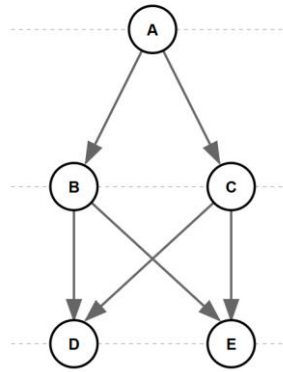


Figure 3-16: Non-level-planar graph.

The Sugiyama framework

The Sugiyama framework allows to generate hierarchical layouts for directed graphs using a 4-step framework. It decomposes the problem of generating a layered visual representation of a directed graph into four modular phases: cycle removal, layer assignment, crossing minimization and coordinate assignment. The framework was introduced in 1981 in the paper *Visual Understanding of Hierarchical System Structures*, by Kozo Sugiyama et al. [2].

Common goals typically motivating the use of the Sugiyama framework include:

- Consistent edge direction (top to bottom)
- Uniform and compact layer distribution (avoid edges spanning over several layers)
- Minimal edge crossings

Unlike tidy tree drawing algorithms, that take as inputs rooted acyclic trees with a single parent per node, the Sugiyama framework handles general directed graphs, including graphs with cycles and nodes of arbitrary in-degree and out-degree. However, many of the core problems addressed in the framework are solved through heuristics, as solutions like crossing minimization or cycle removal are NP-hard, and it is not possible to find exact solutions.

For this reason, the framework might not always yield optimal layouts, some graphs might for example contain crossings in the layout, even if the input graph is theoretically planar. Increasing the number of iterations for some greedy algorithms involved in the framework may sometimes improve results, however this is often not practical, especially for large graphs.

Different implementations of the framework can have significant variations, some might incorporate all the steps in the framework, while others might not (for example, requiring to input a DAG, instead of finding the DAG equivalent of any directed graph), and different algorithms can be used in each phase. However, the generic pipeline that *Sugiyama et al.* proposed and that most implementations integrate is the following:

Step 1: Cycle removal

Also referred to as feedback arc reduction, the objective of this phase is to convert a generic directed graph into a DAG, which is a requirement for next phases of the framework. As discussed in previous sections, finding a minimum feedback arc set is NP-hard, so practical tools rely on $O(E)$ heuristics.

Some implementations of Sugiyama, like Graphviz *dot* algorithm may include a conversion to directed graph before the cycle removal, allowing virtually any graph to be used as an input. On the contrary, there are also some implementations that do not incorporate this step, and therefore require directed acyclic graphs as inputs, like the *SugiyamaLayout* function in Graphistry.

Step 2: Layer Assignment

The objective of this phase is to assign each node to an integer layer, so that edges only go from higher layers to lower layers, (or mostly, if not possible). Objectives include minimizing edges with higher than 1-layer spans, bounding width per layer, or respecting chosen roots and other topological constraints.

Some algorithms that different implementations integrate include, for example, *Longest Path* (which ensures minimal layer count for DAGs), *Network Simplex* (integrated in Graphviz *dot*, minimizes edge lengths) or *Coffman-Graham* (restricts layer width).

Most layer generation algorithms generally do not incorporate the constraint that the resulting layout must be level-planar, generating a level-planar set of layers is an NP-hard problem (as crossing minimization), and detecting if a given graph is level-planar is not as trivial as detecting general planarity. A solution to this problem can be approximated through heuristics, as an exact general solution does not exist [44].

This process might also include dummy node insertion, which helps later steps like crossing minimization by ensuring all edges span over exactly 1 layer, by inserting dummy nodes in between longer spanning edges. Modules like Graphviz *dot* handle this process internally, so that the dummy nodes are inserted before crossing minimization and removed after coordinates are assigned.

Step 3: Crossing minimization

Receiving as an input the layered graph, with the layering assignment created in step 2, this phase of the algorithm has the objective of swapping the relative positions of nodes inside each layer to minimize crossings. Exact minimization of crossings is NP-complete, so a heuristic approach is necessary. Two of the most common methods are the barycenter method and the median method.

The barycenter method consists of iteratively placing each node on the barycenter (mean) X position of its neighbors in adjacent layers. The median method works in the same way, but using median positions instead of average. Generally, several bottom-up and top-down sweeps to optimize node positions. This process spreads nodes naturally and smooths layouts using a simple approach, but is prone to get stuck in local minima, especially for larger graphs, and therefore not be able to minimize all crossings.

A way to improve the results is to introduce a naïve randomization of the input positions, then run the barycenter/median algorithm several times and keep the best solution. This increases the odds that the algorithm finds the optimal solution and is not stuck in local minima, but it

also exponentially increases time complexity of the algorithm when using a large number of iterations.

If the layered graph received as an input is non-level planar, the crossing reduction section will not find a solution with zero crossings, so the effectiveness of the crossing minimization algorithms is always capped when dealing with non-level-planar graphs.

Step 4: Coordinate assignment

Once the relative position of each node in every layer is calculated, each node is assigned an x and y position based on different constraints, like minimal horizontal separation, straight line constraints, or parents centered above successors (like in the Reingold-Tilford algorithm). The original *Visual Understanding of Hierarchical System Structures* paper originally formulated a quadratic program, but more efficient linear-time methods followed and are implemented currently. One of the most popular algorithms is *Brandes & Köpf* (implemented in Graphviz *dot*), which balances edge lengths and straightens long edges.

3.3 Summary

This chapter sets a base for the theoretical concepts that the methodology section relies on. From explaining basic concepts in graph theory, to graph traversal algorithms that constitute the base for several algorithms used in this project's methodology, the definitions explained in this chapter are largely referred to over the remainder of the thesis.

Building on that foundation, the chapter also explains graph layouts, and provides an in-depth examination of two key algorithms, the Reingold-Tilford algorithm, simple and most suitable for creating rooted tree layouts, and the more general, Sugiyama approach, that over its 4-step process can generate hierarchical layouts for more complex graphs. These layout algorithms are applied in the methodology section and constitute an essential step in the generation of single-line diagrams.

4 Methodology and Implementation

4.1 Formal Problem Framing

The automatic generation of single-line diagrams from GIS data can be formally defined as a graph transformation and visualization problem. The input network is defined as a spatial network graph $G = (V, E, P, A)$ where:

- $V = \{v_1, v_2, \dots, v_n\}$ is the set of nodes representing network components.
- E is a set of edges representing electrical connections between elements.
- $P: V \rightarrow \mathbb{R}^2$ is a position function mapping each node to geographic coordinates.
- $A: V$ is an attribute function mapping each node to a set of properties. Among these properties, is the component type, $type_i$. The set of all entity types in a graph is defined as T .

The output is a diagram representation $S = (G', L, R)$, where:

- $G' = (V', E')$ is a simplified directed acyclic graph derived from G .
- $L: V' \rightarrow \mathbb{Z}^2$ is a layout function mapping each node to a discrete grid position.
- $R: V' \rightarrow S$ is a rendering function mapping nodes to diagram symbols

4.1.2 Input Specification

The algorithm takes as an input a GIS data structure containing a section of the LV distribution network section in graph format (nodes and edges) and a set of configuration parameters, that allow to adjust some desired characteristics in the final diagram.

The network data file must contain:

- Set of nodes $\{v_i\}$ representing elements and components in the network, with properties:
 - Unique identifier id_i
 - Component type: $type_i$
 - Other electrical properties: $props_i$
- Set of edges $\{e_j\}$ either directed or undirected, with properties:
 - Source node: u_j

- Target node: v_j

Configuration parameters:

- $\tau_{\text{key}} \subseteq T$: Set of key component types to avoid simplifying and therefore preserve in the final graph G' .
- ρ_{root} : entity type label that identifies root nodes.
- λ_{max} : Maximum consumers per bus threshold.
- γ_{group} : Maximum consumers per group.
- $iter_{\text{max}}$: Maximum layout algorithm iterations.

Each configuration parameter and its effect on the final diagram will be further explained over the following sections.

4.1.3 Output Specification

The algorithm generates:

1. Schematic Data Structure: CSV file containing:
 - Discrete node positions: $(x'_i, y'_i) \in \mathbb{Z}^2$ for each $v_i \in G'$
 - Node types and rendering properties
 - Size values for buses and connectors
2. Visual Representation: SVG file containing a render of the data structure with:
 - Orthogonal layout with discrete grid spacing
 - Standard electrical symbols for each component type
 - Hierarchical structure with clear bus-branch relationships

4.1.4 Constraints and objectives

The solution must satisfy the following invariant constraints and objectives:

1. Topology Preservation: for every pair of nodes in the simplified graph G' , their connectivity must be preserved from the original network G :

$$\forall u', v' \in V': \text{path}(u, v) \in G \Rightarrow \exists \text{path}(u', v') \in G'$$

2. Hierarchical Structure: The output graph must have a layered layout $L: V'$ that meets:
 - Elements with $\text{type}_i = \rho_{\text{root}}$ are placed on the first layer
 - Buses can have multiple predecessors and successors; bus successors of each bus will be strictly placed on lower layers.
 - The buses are connected with each other through vertical lines.

3. Key Component Preservation: All components in τ_{key} must appear in the final diagram:

$$\forall v \in V, (\text{type}(v) \in \tau_{\text{key}}) \Rightarrow \exists v' \in V'$$

4. Orthogonality: All connections in L must be horizontal or vertical
5. Crossing-free: the solution must minimize crossings in its underlying graph layout.

$$\min_L C(L) = \sum_{(e_1, e_2) \in E' \times E', e_1 < e_2} 1 [\text{crosses}_L(e_1, e_2)]$$

Although the algorithm treats this as a crossing minimization problem, the result will be considered invalid if the following constraint is not satisfied:

$$\forall e_i \neq e_j \in E': e_i \cap e_j = \emptyset$$

6. Simplified representation: graph G' must contain a smaller number of nodes than graph G :

$$|V'| < |V|$$

4.2 Solution Overview

4.2.1 Schematic structure and requirements

The proposed solution generates orthogonal topological diagrams that follow standard electrical diagrams conventions, while ensuring clarity and readability. The output diagram has a hierarchical structure that allows to reflect the electrical distance and relationships between different buses. The diagram is organized as a layered graph where:

- Horizontal lines represent buses, electrically equivalent connection points.
- Vertical lines represent connections between buses at different.
- Standard symbols (which are also laid out vertically) represent key network components, like transformers, switches, fuses, or consumers.

The topmost layer contains the elements identified with parameter ρ_{root} , which in the case of LV networks corresponds to distribution transformers. Subsequent layers show the distribution of buses in the network, and the elements connected to each, representing increased electrical distance from the root(s) for lower layers.

This hierarchical representation achieves several visualization goals:

- Displaying electrical equivalence: elements connected to the same bus are easily identifiable as elements being connected to equivalent electrical points. The criteria to determine electrical equivalence is further explained in the next section.
- Layers showing electrical distance: the vertical layering allows displaying the relative electrical distance between different elements in the network to the distribution transformer, providing an intuitive view of the network's power flow hierarchy.
- Special component relationship: the diagram depicts connectivity relations between elements in the network and special nodes like switches or fuses, through their relative positions and buses, allowing for example to clearly visualize which consumers get connected or disconnected when activating a switch.

To achieve this structured visualization, the underlying graph must conform to a specific normalized pattern before the layout phase. This normalized structure ensures that:

- Every connection between buses is represented by exactly one node, which can be a simple line or a specific network element, like a bus or a switch.
- Bus nodes are therefore alternated in a regular pattern: bus → connection → bus nodes. Bus nodes can have additional successors, or two connection node predecessors, but buses cannot.
- No bus nodes are adjacent.
- Each bus in the graph corresponds to exactly one diagram symbol.

This normalized structure is essential because the diagram plotting module operates by:

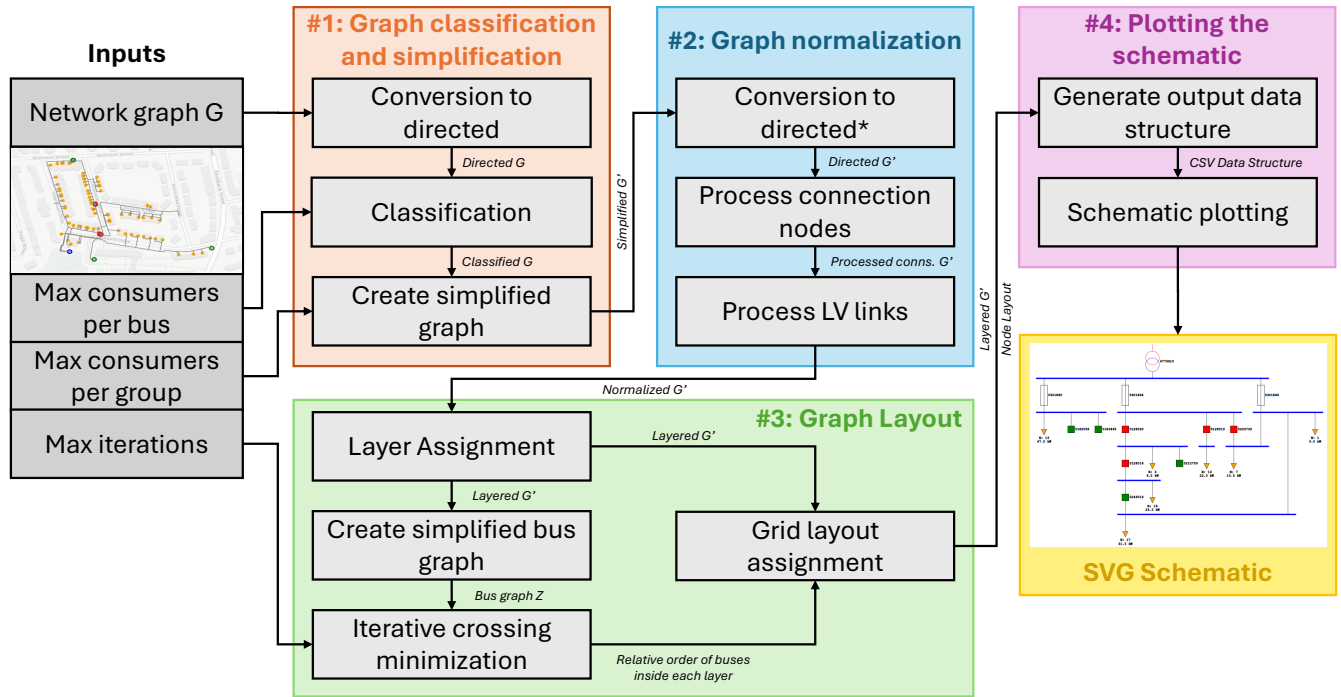
- Taking the nodes position as the top-most reference point.
- Assigning each diagram symbol based on the node's properties (mainly type).
- Drawing connections implicitly through symbol placement, the module does not render edges.

Therefore, the graph transformation phase must ensure that the graph meets this structure before the layout algorithm assigns each node's positions, making normalization a critical requirement for successful diagram generation.

Additionally, the resulting output must provide a simplified view of the network, helping visualize more clearly its topology than the underlying GIS representation, where excessive detail and large amounts of cluttered elements can obscure the essential connectivity and topological relationships. To achieve this, a set of classification and simplification rules is applied to ensure that the diagram output is a faithful but simplified representation of the underlying network.

4.2.2 High level solution architecture

The solution proposed takes a 4-step process to generate the final diagram representation from the underlying raw GIS data. This process can be observed in Figure 4.1:



**Second conversion to directed considers special rule of predecessors to bus nodes*

Figure 4-1: Methodology diagram.

Justification:

The four-phase process (classification → normalization → layout → rendering) is adopted for six main reasons.

1. Reduced complexity: handling tasks sequentially is fundamental to ensure all objectives are satisfied by evaluating each of them by their individual output. Dividing the task into separate phases lets each phase isolate one objective while preserving constraints; pursuing all objectives simultaneously would be impractical and inefficient.
2. Scalable simplification. Real GIS data sections of LV/MV networks contain thousands of individual points. Grouping consumer nodes and connection points that reflect non-essential junctions can significantly reduce the node count of the networks being processed (by up to x10 to x20 times, for the cases studied), significantly improving the algorithm's efficiency.
3. Layout-aware normalization. The layout must follow a strict set of constraints, as described before, which rely on the underlying structure of the graph. The normalization process enforces

a bus \rightarrow connection \rightarrow bus node structure and converts cycles into DAG structures thereby ensuring compatibility with a standard layout generation engine that does not need to deal with several particular cases.

4. Iterative crossing minimization. Inside the layout phase, the crossing minimization algorithm is iteratively run to maximize the probability of finding the optimal result. This process seeks to increase the odds of finding the optimal solution by broadening the search space via randomization. Feeding the algorithm a simplified graph with only the bus nodes information is an essential step in reducing the computational complexity of this iterative process.

5. Layout assignment: the layout assignment phase assigns the coordinates to each node satisfying the constraints and objectives defined earlier. Taking as an input separately the relative order of buses in each layer, and the normalized graph to be laid out ensures it can focus on geometric placement without having to optimize all topological and crossing minimal requirements simultaneously.

6. Modularity and versatility. Task separation also allows for a modular algorithmic pipeline, in which specific sections can be replaced to improve performance, handle data with different characteristics, or introduce small adjustments to tune the results. For example, decoupling positioning from symbol rendering allows integration with different applications by adapting styles, or output formats, without modifying the algorithmic base.

4.3 Implementation Details

4.3.1 Development language and dependencies

The language chosen to implement the pipeline is Python 3.13.1 [45], due to its versatility and wide range of libraries and pre-existing implementations, as well as to ensure compatibility with other SPEN network connectivity applications. Additionally, the following libraries and dependencies required:

Core logic libraries:

- Networkx 3.4.2: wide range of graph algorithm and functions required for the logic of the script, conveniently handling graph as classes, and performing operations like neighbour checks [46].
- Pandas 2.2.3: used for handling several data structures as *dataframes* [47].
- Graphistry 0.39.1: supplies a pre-built implementation of the barycentre heuristic used in the crossing minimization step, through the *SugiyamaLayout.arrange()* function [48].
- Numpy 2.2.0: several useful mathematical functions [49].

Visualization libraries:

- Geopandas 1.1.1: used for handling geographical coordinates in *dataframes* to create GIS visualizations [50].
- Folium 0.20.0: generates visualizations of GIS data, overlaid on maps [51].
- Plotly 5.24.1: used to create visualizations of graphs [52].
- Graphviz 0.20.3: generates the graph layouts for visualization (used in creating the Figures, not the actual diagram layout) [53].
- Svgwrite 1.4.3: used to generate the output rendered diagram on SVG format [54].

4.3.2 Data characteristics

Although the proposed methodology constitutes a pipeline that can be used to generate topological diagrams for a wide range of input data structures, each with their own particular characteristics, in this project it has been implemented to produce diagrams for *SPEN* LV distribution grids data in particular.

The input data used for the implementation is read from GeoJSON files exported directly from *SPEN's* GIS systems. These files contain LV network sections in a graph format, with nodes representing different network elements and junctions between cables; and edges representing the cables connecting those nodes.

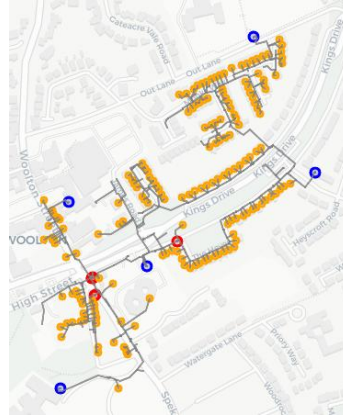


Figure 4-2: Example of an input LV network section.

The list of properties contained in the raw data that have been used in the implementation can be found in Table 4.1. These properties serve several purposes, some are crucial for the algorithm's logic, while others just reflect relevant information to be displayed in the final diagram.

Table 4-1: Properties used in the implementation.

id	Unique identifier for each node, mapped to the already defined id_i
entity_type	Indicates the type of element for each node, mapped to the already defined $type_i$
admd_max_sum	Total contracted demand for each node in kW. Mapped for aesthetic representation in the final diagram
circuit_id	Identifier for every node that makes up each of a transformer's feeders. Mapped for aesthetic representation in the final diagram

Relevant entity types and their labels in the data include:

- **Dist_Transformer**: MV/LV distribution transformer.

- LV_MSP: consumer(s).
- LV_Link: low voltage link boxes.
- LV_Fuse: fuses, connected downstream from distribution transformers.
- LV_Joint: intersection between two cables.

For representing LV network sections, the set of Key Nodes and root node label are therefore assigned to:

$$\tau_{\text{key}} = \{\text{Dist_Transformer}, \text{LV_Link}, \text{LV_Fuse}\}$$

$$\rho_{\text{root}} = \text{Dist_Transformer}$$

LV links are nodes which constitute the terminals of LV link boxes, a LV element present in SPEN distribution networks, which have the role of rerouting feeders in LV grids, enabling maintenance works and service restoration. These nodes can be connected or isolated, and to ensure that the resulting diagram represents the connectivity properties faithfully for every LV link-box, a specific part of the process deals with these nodes in particular.

Consumers are identified as nodes with entity type = LV_MSP, which reflects service points to which one or several consumers are connected.

4.4 Model Development

This section explains the methodology and implementation process in detail, explaining what each step in the algorithm aims achieve, as well as the rationale for its inclusion in the final pipeline. As mentioned before, the implementation has been tailored to the input data used, *ScottishPower Energy Networks* LV feeders, so some specific steps in the process might not be strictly necessary for implementations using data with different characteristics, while they might require specific adjustments or individual steps as well.

For consistency among the current implementation explanation and the code and pseudocode sections, the parameters and variables mathematically defined in the previous sections will be referred to with the following labels and variable names:

- ρ_{root} : *root_label*.
- λ_{max} : *max_consumers_per_bus*.
- γ_{group} : *max_consumers_per_group*.
- τ_{key} : *key_labels*.
- $iter_{\text{max}}$: *max_iters*.
- id_i : *node_id*.
- $type_i$: *entity_type*

Additionally, the legend for the graph figures used in this section can be observed in Figure 4.3. The node types are further explained in the classification section.





	Bus node, represents horizontal electrically equivalent lines (buses) in the final diagram.
	Connection node, represents vertical lines connecting different buses in the final diagram.
	Consumer node, represents consumers in the final diagram. Before simplification, they also represent leaf nodes, non-consumer nodes with degree 1.
	Key node, represents relevant grid components in the final diagram. This category can include transformers, fuses or LV link boxes, for example.

Figure 4-3: Node legend for graph figures

4.4.1 Graph classification and simplification

Inputs: Raw input graph G , and parameters $max_consumers_per_bus$, $max_consumers_per_group$. Internally, for the current implementation, set of labels key_labels and $root_label$.

Output: classified and simplified graph G' .

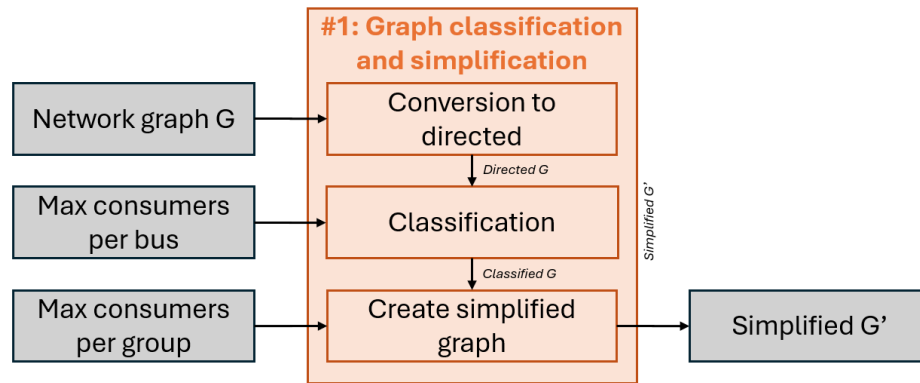


Figure 4-4: Classification and simplification phase.

The goal of the classification and simplification phase is to create a simplified subgraph G' from the input graph G , reducing the number of nodes and grouping consumer nodes so that the resulting single line diagram representation contains only the key topological information of the underlying network. This process allows to reduce visual clutter, while also ensuring that the key connectivity information between all the elements in the network is preserved.

The classification rules define which nodes should be retained in the final diagram and which serve only connectivity purposes and can therefore be simplified to reduce visual clutter. This decision is based primarily on each node's *entity_type* property, its number successors and their respective classifications. The classification process assigns a “role” to each node, reflecting its function within the network whether it represents an electrically equivalent point (to be grouped into a single bus), a connection point between different buses, or a critical element such as a transformer or switch that must be preserved. Determining what elements are connected on the same bus relies on calculating the number of consumers a resulting bus would connect, if said number is higher than the input parameter $max_consumers_per_bus$, they are separated, so that in the final diagram they are represented as different buses. Furthermore, nodes with *entity_type*

labels contained in *key_nodes*, are considered essential elements that must be represented in the final diagram and are therefore not simplified.

Given this classification, a simplification algorithm then creates groups, so that electrically equivalent points are represented with a single node (bus) in the final graph, and consumers connected to the same bus are also grouped, while the nodes that represent important elements in the original graph are kept. The resulting graph contains a significantly lower node count than the original graph and is therefore more suitable for representation in a topological diagram.

Converting the graph to directed

The classification algorithm runs from the leaves (single degree nodes) to the roots of the graph (distribution transformers for LV sections), as the classification of each node depends on the classification of its successors. However, the input graph is undirected, so to simplify the logic of the classification algorithm it is simpler to first convert the graph to directed and rely on the predecessor/successor logic to apply the classification rules. If the input graph is already directed, in a direction that follows the power supply path from transformers to consumers, this step is not necessary.

The conversion to directed is simple, the algorithm takes as an input the root nodes (distribution transformers in the case of LV grids), and starting from these nodes, traverses the whole graph through BFS, converting the edges to directed pointing in the order of the traversal.

This generates a directed acyclic graph, to which the classification rules can be applied.

Node classification

The classification algorithm assigns the role to each node in the graph, which determines how it is simplified and laid out in the following sections. As mentioned before, the algorithm determines whether each node is a bus node (adjacent bus nodes are considered electrically equivalent, and will therefore be grouped into a single node later), a connection node, that connects two or more buses, a consumer node (will be grouped with adjacent consumer nodes in the same graph), or a *key* node (will be represented in the simplified graph). The characteristics and criteria for classification are the following:

- **Consumer nodes:** these nodes represent the connection points for clients serviced by the distribution grid section and are identified using their entity type property. In the flagging process, they are also identified as part of a broader category, leaf nodes, a classification applied to every single-degree node (except for key nodes, explained below), and two-degree nodes connected to a leaf node (or nodes with n degree but $n-1$ leaf node neighbors). This leaf node category is necessary to correctly perform the classification algorithm and identify bus nodes.
- **Bus nodes:** represent electrically equivalent points in the network, that are represented as buses in the final diagram and serve as the connection points for consumers and other relevant elements in the network. These nodes are grouped with adjacent bus nodes by the simplification algorithm, to allow obtaining a faithful but less cluttered topological representation of the original network.
- **Connection nodes:** these nodes connect the different buses in the network; their flagging is required to mark the boundaries for different electrically equivalent zones. They also ensure that the key topological features of the original network are preserved, for example main joints in the original graph which may connect several different main feeders in the real network are preserved by classifying nodes adjacent to three or more bus nodes as connection nodes.
- **Key nodes:** these nodes represent key elements in the original network, that must be represented in the simplified graph and not simplified. They are identified by an input set of labels that are collected beforehand. For the LV networks studied, this includes transformers, switches, fuses and LV links (will be further explained in following sections).

The criteria to determine whether adjacent nodes should be considered electrically equivalent, and therefore part of the same bus, is determined by the *consumer_count*, i.e. the number of consumers that are connected to the same bus (adjacent bus nodes are part of the same bus). In addition to the graph, the algorithm takes as an input the *max_consumers_per_bus* parameter, which determines the maximum number of consumers that adjacent bus nodes can have to be considered electrically equivalent. It also takes as an input the consumer label, as consumer nodes are identified by their entity_type property (in this case, *LV_MSP*, as explained earlier).

Algorithm 4.1: CLASSIFY_NODES (simplified)

```

Input: Rooted DAG  $G = (V, E)$ ; consumer_label;
        max_consumers_per_bus; key_label
Output:  $G$  with every node flagged leaf_node, bus_node or
        connection_node (mutually exclusive) and key_node

// 1. Leaf pass
1 foreach  $v : DEG[v] = 1$  do
2   if  $v.entity\_type = key\_label$  then mark  $v$  key_node; continue;
3   mark  $v$  leaf_node; if  $v.entity\_type = consumer\_label$  then
    $v.consumer\_count \leftarrow true$ ;  $v.consumer\_count \leftarrow 1$ ;

// 2. Interior pass (reverse topological order)
4 for  $v$  in reversed(topological_order)(G) do
5   if  $v$  already classified then continue;
6    $succs \leftarrow successors(v)$ ;  $preds \leftarrow predecessors(v)$ ;
    $n\_demand \leftarrow |\{u \in succs \cup preds \mid u.leaf\_node\}|$ ;
    $n\_non\_demand \leftarrow DEG[v] - n\_demand$ ;
    $sum\_cons \leftarrow \sum_{u \in succs} u.consumer\_count$ ;
7   if  $v.entity\_type = key\_label$  then
8     mark  $v$  key_node; if  $DEG[v] \geq 2$  then mark  $v$  connection_node;
      $total\_consumer\_count += sum\_cons$ ; continue;
9   if  $DEG[v] = 2 \wedge |succs| = 1 \wedge succs[0].leaf\_node$  then
10    mark  $v$  leaf_node;  $v.consumer\_count \leftarrow$ 
      $sum\_cons + (v.entity\_type = consumer\_label)$ 
11  else if  $n\_demand = DEG[v] - 1$  then
12    mark  $v$  (if ( $sum\_cons < max\_consumers\_per\_bus$ ) : leaf_node else :
     bus_node);  $v.consumer\_count \leftarrow sum\_cons$ 
13  else if  $n\_demand = DEG[v] - 2$  then
14    mark  $v$  bus_node;  $v.consumer\_count \leftarrow sum\_cons$ 
15  else if  $n\_non\_demand > 2$  then
16    mark  $v$  connection_node;  $total\_consumer\_count += sum\_cons$ 
17  else
18    mark  $v$  connection_node
19  if  $v.bus\_node \wedge v.consumer\_count > max\_consumers\_per\_bus$  then
     mark  $v$  connection_node;  $v.consumer\_count \leftarrow 0$ ;
20 return  $G$ ;

```

A simplified pseudocode describing the high-level logic of the classification process can be observed in Algorithm 4.1. The algorithm has a time complexity of $O(2 * n + e)$, as it does two traversals of the nodes in G (one through the leaves and another on the complete graph), and a single neighbor check, for every node in the graph, adding a complexity equal to the number of edges.

This classification ensures several requirements. First, bus nodes are identified as electrically equivalent nodes adjacent to each other, based on whether their total consumer count is not

higher than the threshold specified. Second, connection nodes mark the boundaries for these electrically equivalent buses. Third, key nodes are correctly flagged, so that the simplification algorithm can identify them and not group or remove any of them.

Groups of consumers bigger than the *max_consumers_per_bus* parameter can be connected to the same bus, but only if in the original graph they were connected at the same node (same electrical level), as the algorithm does not change this property and it will remain in the simplified graph. However, with a similar parameter used in the graph simplification algorithm, the maximum sizes of the consumer groups inside each bus can also be adjusted.

Creating simplified graph

Taking as an input the classified graph, the simplification algorithm creates a new graph that contains a reduced number of nodes, by grouping bus nodes and consumer nodes. This process is performed based on the rules outlined in the previous section, to ensure that the resulting graph contains faithful topological information about the underlying real network, yet with a reduced number of nodes.

Connection and key nodes are added to the simplified graph directly, with the same properties as they had in the complete graph. Then, adjacent bus nodes are grouped into a single node and connected to the intersection nodes that the boundary bus nodes of the group were connected to originally. The consumer nodes adjacent to any of the grouped bus nodes are also grouped, in sizes equal to an input variable called *max_consumers_per_group*. Consumer nodes that are directly connected to connection nodes in the complete graph, also get grouped with adjacent consumer nodes and connected to their respective connection node in the simplified graph.

The input variable *max_consumers_per_group* determines the maximum number of customers to group for every adjacent set of consumers. This parameter ensures that the size of the consumer groups does not exceed a determined value, and along with the *max_consumers_per_bus* variable provided in the classification algorithm, allows to adjust the level of detail that the graph displays about the underlying real network.

If the *max_consumers_per_group* parameter is set equal to the *max_consumers_per_bus* variable, each bus node will generally only have one group of consumers, with size equal to both. However, if a smaller size for *max_consumers_per_group* is used, several groups of

consumers will be created per bus node. Special situations can occur where a bus has a higher number of consumers that the *max_consumers_per_bus* parameter, which are caused by several consumers connected to the same node in the complete graph, so if it is desired that only one group of consumers is connected to each bus node, a higher *max_consumers_per_group* should be used. The effect of adjusting these parameters on the resulting diagram is explored in the case studies section.

The edges created to generate the simplified graph do not consider the original direction of the edges in the complete graph and are added as undirected edges. This is done for two main reasons:

1. Checking for directionality for every node in the complete graph implies a computational time complexity proportional to the size (n nodes) of the graph, $O(n)$, however converting the graph to directed once it has been simplified requires a single traversal of the graph through BFS, which also has $O(n)$ time complexity, but the simplified graph has fewer nodes, so it is more efficient to convert the graph to directed again after the simplification.
2. The logic for directing the graph after its nodes have been classified and simplified is not the same, as after the classification, only bus nodes will be allowed to have more than two predecessors. The logic and reasoning behind this will be further explained in the following sections.

Figure 4.3 contains a graphical representation of a classified graph's section, input to the algorithm, the simplified graph G' output. The key node in Figure 4.4 is also classified as a connection node, as its degree is equal to 2. As it can be observed, the algorithm groups all the adjacent bus nodes, and the consumer nodes connected to them, and connects the resulting bus node group to the connection nodes that the original group of bus nodes were connected to.

(a)

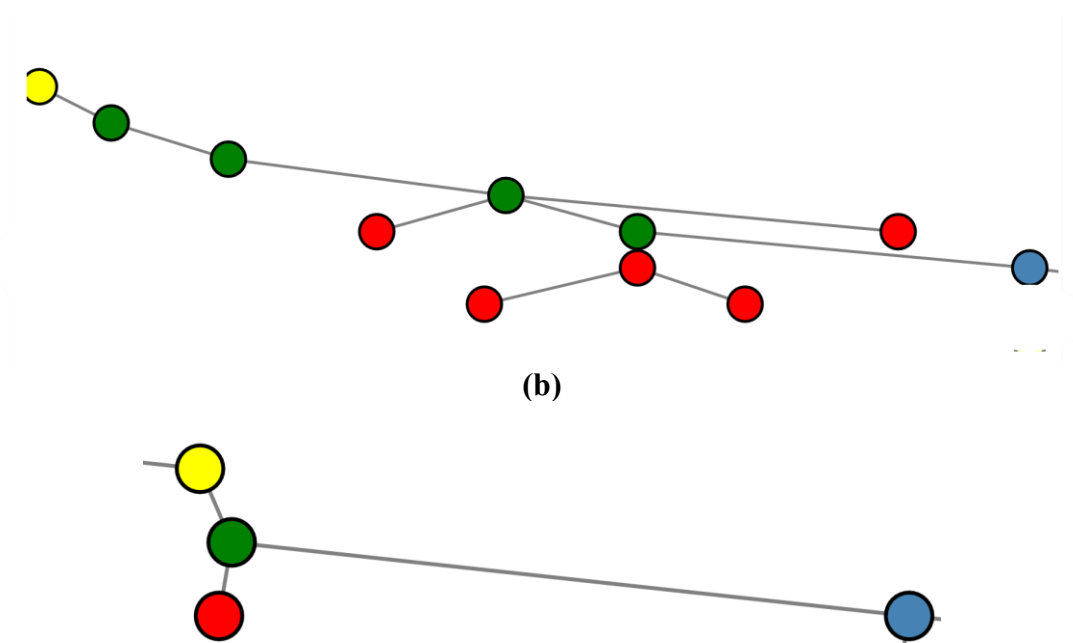


Figure 4-5: (a) Classified graph G input (b) Simplified graph G' , output of the simplification algorithm.

Once the simplified graph has been generated, it must be processed because it may contain some undesirable structures in the graph, like several connection nodes adjacent to each other, or consumer nodes adjacent to connection nodes directly; and, as mentioned before, the graph must have a normalized structure of bus-connection-bus nodes to be laid out and subsequently plotted.

4.4.2 Graph normalization

Input: Simplified graph G' .

Output: Normalized graph G' , bus-connection-bus nodes.

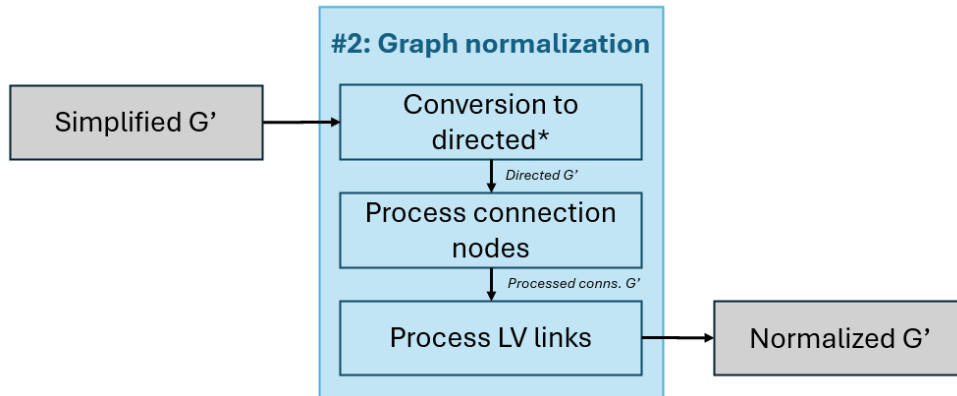


Figure 4-6: Graph normalization phase.

The objective of this phase is to ensure the simplified graph follows the normalized bus-connection-bus node structure that is required by the layout algorithm and to plot the final diagram. The steps involved in the normalization process are complementary to the classification and simplification process, as the node classification sets the base for the graph structure, and the normalization phase simply ensures this structure is consistent.

This need for normalization arises from the specific requirements and constraints imposed about maintaining specific elements from the complete network graph (key nodes) and some connection nodes, and the need to generate an structure that can actually be laid out in the shape of an diagram and retains the topological characteristics of the network. These requirements prevent the classification algorithm to generate a structure that complies with the bus-connection-bus requirements as, for example, two key nodes might be adjacent.

The post processing algorithm takes as an input the undirected simplified graph and transforms it into a DAG that meets that all its nodes follow the bus-connection-bus node structure explained earlier. To ensure this, several specific scenarios need to be considered, and specific repeating structures in the graph addressed, which include: several adjacent connection nodes, key nodes connected directly to each other, and demand nodes connected directly to connection

nodes, among others. It is ensured that the graph resulting from the transformations in this section meets that:

- Connection nodes have only one predecessor and one successor, both bus nodes.
- There are no bus nodes adjacent to each other.
- Demand and key nodes are successors to bus nodes. Connection key nodes have a predecessor bus node and a successor bus node.

Meeting these conditions is necessary to ensure the graph can be successfully laid out and plotted.

Another requirement is that the normalized structure must be achieved without modifying the basic underlying topology of the network. All the rules and algorithms applied ensure that the nodes retain their relative position with the key nodes in the original graph, and that the buses created in the classification and simplification section are maintained, so that no consumers are moved upstream or downstream after the post-processing.

Note: this section is largely adjusted to the characteristics of SPEN's data. Using networks with different structures may require adjusting some of the algorithms applied in this phase. However, the general pipeline of removing adjacent connection nodes, and inserting bus nodes where required should be applicable to any given starting graph, while retaining the essential topological characteristics.

Re-conversion to directed

As mentioned in the graph simplification section, the simplified output graph is undirected, as making it directed while it is simplified would further complicate the logic for that function, while converting the graph to directed once it has been completely simplified requires just a single traversal of the network.

The function to convert the graph to directed is the same as the one used in the previous section, but with an additional consideration, the requirement that only bus nodes can have more than one predecessor.

To ensure this, during the BFS process the algorithm must consider whether the node to connect downstream already has a predecessor. If it does, then the node type is checked, if it is a feeder

node then the algorithm continues directing the rest of the graph as normally, but if it not, either of the preceding edges to that node must be reversed. If either of the preceding nodes is a bus node, one of the edges is simply reversed, but if neither of them is, edges are reversed upstream, up to the previous bus node. During this process, it is necessary to ensure that no cycles are created, if reversing either of the paths creates a cycle, the other predecessor's path is reversed instead.

After this process, it is ensured that only bus nodes have more than one predecessor, so every key node and bus node have a single predecessor. After all the edges have been redirected, the resulting directed acyclic graph can be processed by next step, which consists in processing connection nodes.

Processing connection nodes

The simplification algorithm takes as a base the connection nodes in the complete graph to recreate the simplified version of the complete graph, as these nodes preserve the core topology information about the real network. Therefore, any adjacent connection nodes in the original complete graph would result in adjacent connection nodes in the simplified graph. This is not desirable, as the structure of the graph must follow the bus-connection-bus node explained earlier, so adjacent connection nodes must be processed to ensure that this does not occur in the final graph.

Additionally, connection nodes are guaranteed to only have one predecessor at this stage (as described in the converting to directed section), though they might have several successors. In the final normalized structure, however, every connection node must be linked to exactly one upstream and one downstream bus node.

To solve both problems, a two-step algorithm is applied. The first step consists in “splitting” connection nodes, by creating a copy of the original connection node for every successor. The resulting copies are each connected to the original predecessor of the connection node, and to one of its successors. The original node is removed, but each of the copies has the same properties as it had.

In the second step, every connection node that has either a connection node predecessor or successor is removed. The successor and predecessor of the removed node are then joined. Each

of these steps is applied recursively, to ensure that no multi-successor or adjacent connection nodes remain. The complete high-level process is outlined in Algorithm 4.2.

This two-step algorithm is computationally simple and generalizable, as it allows to simplify a wide range of scenarios effectively without specifically creating a matching algorithm for every particular case. For instance, if the adjacent connection nodes removal process was done independently before splitting the connection nodes, the successors to each connection node to be removed would have to be checked to ensure that after removing the connection node, no restrictions like connecting directly two bus nodes would be violated. By first ensuring that every connection node only has a single predecessor and successor node, the logic is significantly simplified. Also, since connection nodes would still have to be split afterwards, this algorithmic order minimizes complexity and allows generalization.

Non-single-degree key nodes, which are also flagged as connection nodes, are not modified by this algorithm, as creating copies of them would incorrectly reflect non-existing elements in the final diagram, and removing any of them would violate one of the key requirements of this process, which is to preserve the topological accuracy of the resulting diagram.

Algorithm 4.2: PROCESS_CONNECTION_NODES

Input: Directed simplified graph G'
Output: Graph G' with normalized connection nodes (non-key), each having one predecessor, one successor and with no adjacent connection nodes

// PHASE A: Split connection nodes

```

1 repeat
2    $C \leftarrow \{c \in G \mid c.\text{connection\_node} \wedge \neg c.\text{key\_node} \wedge \text{outdeg}(c) > 1\};$ 
3   if  $C$  is empty then
4     break
5   foreach  $c \in C$  do
6      $p \leftarrow \text{predecessor}(c);$ 
7     foreach  $s \in \text{successors}(c)$  do
8        $c' \leftarrow \text{clone}(c);$ 
9       add edge  $p \rightarrow c'$  (attrs =  $p \rightarrow c$ );
10      add edge  $c' \rightarrow s$  (attrs =  $c \rightarrow s$ );
11      remove edge  $c \rightarrow s$ ;
12 until no connection node with  $\text{outdeg} > 1$ ;

// PHASE B: Remove adjacent connection nodes
13 foreach  $n$  connection node with
     $\text{indeg}(n) = 1 \wedge \text{outdeg}(n) = 1 \wedge \neg n.\text{key\_node}$  do
14    $p \leftarrow \text{predecessor}(n); s \leftarrow \text{successor}(n);$ 
15   if  $p.\text{key\_node} \wedge s.\text{bus\_node}$  then continue;
16   if  $p.\text{connection\_node} \vee s.\text{connection\_node}$  then
17     add/merge edge  $p \rightarrow s$  (attrs =  $n \rightarrow s$ );
18     remove node  $n$ ;

// Tidy-up: single-degree connection nodes
19 foreach  $c$  connection node with  $\text{degree}(c) = 1$  do
20   if  $c.\text{key\_node}$  then  $c.\text{connection\_node} = \text{False}$ ;
21   else remove  $c$ ;
22 return  $G$ ;
```

Processing LV Links

This function deals with the specific LV link nodes present in *SPEN's* data, so this section of the methodology is unique to this project's implementation. However, it can be used to deal with network elements with similar characteristics, like switches or breakers, where the two terminals of a single element are represented by two separate nodes.

The basic logic for processing LV links is very similar to the logic for processing other connection nodes. However, there are several considerations specific to LV Links that require that they are processed independently:

- First, processing them as regular connection nodes could cause some LV Links to be deleted in the adjacent connection nodes removal process.
- Second, LV link nodes are not split if they have more than one successor, as this would lead to incorrectly representing more LV links in the final diagram that actually exist in reality. This particular case is explained in the following section, creating synthetic buses
- Third, LV link nodes that represent an open LV link in real life are reflected in the graphs as independent, disconnected nodes. However, as in reality they are part of the same link box, a dedicated algorithm checks for these cases and joins the nodes so that they can be represented as a single element in the final diagram, showing which two buses they would join if they were closed.

LV links can be opened or closed, and that property can be inferred by their degree in the graph. If they are single degree nodes it means the LV link is open, as it does not connect different elements in the network. If their degree is higher, it means that they are closed.

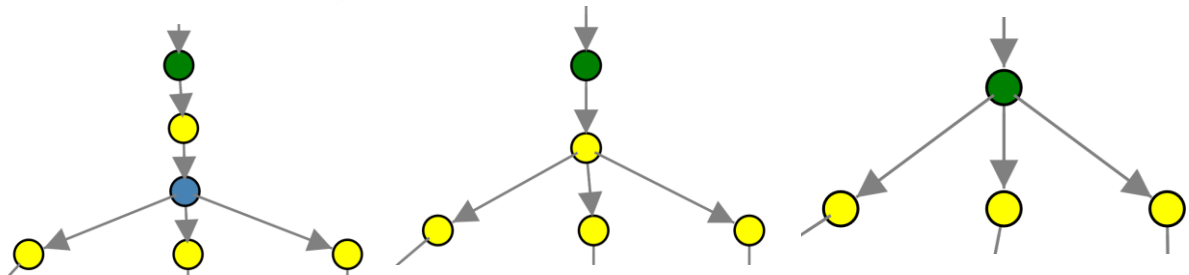


Figure 4-7: Connected LV links normalization (yellow nodes).

In the original data, no LV link has a higher degree than 2, and closed LV links are represented by several LV link nodes adjacent through a connection node. Therefore, after processing the connection nodes, upstream LV links will be connected to several other LV link nodes, as shown in Figure 4.7. These adjacent LV links must then be simplified to represent them as a single node, necessary for the layout logic, that requires that LV links are connection nodes with a bus node upstream and another one downstream.

These nodes are processed like regular connection nodes, the upstream link node is removed, and the downstream link nodes are connected to the bus node that the upstream node was connected to (if the upstream node was an intersection node it would have been removed by the connection nodes processing). A property is added to the remaining nodes, that reflects that they represent closed LV links. In the rare case that there are non-LV link nodes connected to the upstream LV link (which could be caused by non-LV link nodes being connected to the original intersection node), the upstream LV link node is not removed, and instead it is connected only to the remaining non-LV link nodes.

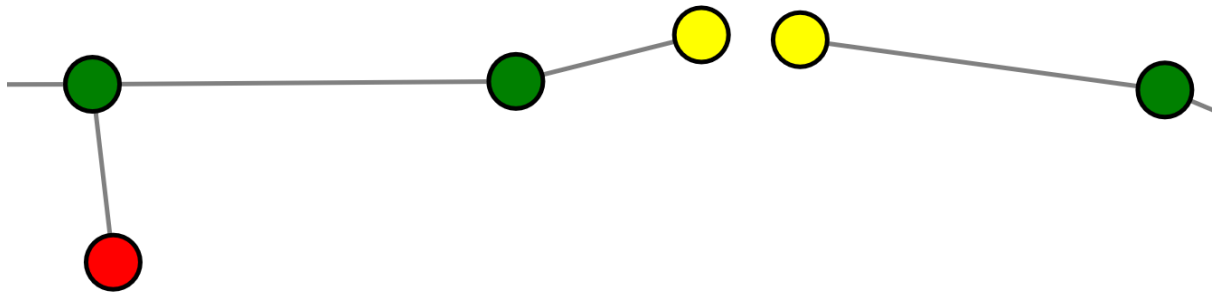


Figure 4-8: Disconnected adjacent LV link nodes.

Unconnected single degree LV link nodes can either be standalone, or adjacent nodes that are part of the same LV link but are disconnected and therefore represented as independent nodes in the graph. As a result, it is not possible to determine whether such nodes are part of the same LV link just based solely on their neighbors, but it is desirable that the diagram represents that relationship, by reflecting what bus nodes would the LV link connect if it were closed.

To identify LV link nodes that are part of the same element even if they are disconnected, it is necessary to perform a search by the node ids. Adjacent LV link nodes have consequent node ids even if they are not connected, so this property can be leveraged by checking whether other standalone LV link nodes have IDs adjacent to a given standalone node.

If a match is found, a distance algorithm is used to check the minimum distance between the nodes in the graph, and it is checked whether that distance is higher than 2 or 3 nodes. If it is, an edge is added connecting both LV link nodes. The distance check is done to avoid adding an

extra edge to single degree LV link nodes that are already connected. Figure 4.9 shows an example where this case would happen.

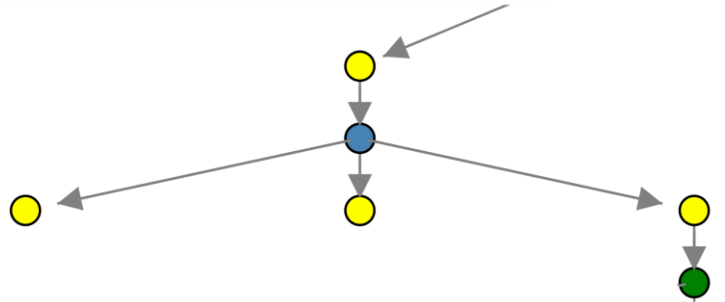


Figure 4-9: Single degree LV link nodes with adjacent IDs.

If the input graph corresponds to a fragmented network (two or more unconnected subgraphs), and the adjacent id search algorithm returns a pair of LV link nodes that return an error to the distance search function because no path exists, this implies that the pair of LV links represent a link in the real network between their unconnected sections. Therefore, these nodes can be joined, and the unconnected subgraphs are converted into a single graph.

After two disconnected LV link nodes are joined, one of the nodes is removed like all regular connected LV link nodes are, and the resulting node, which connects the two bus nodes that were adjacent to each standalone node, is flagged as a connection node.

Creating and processing bus nodes

Depending on the topological characteristics of the network section being represented, mainly caused by the placement of key nodes and nodes with degree > 3 inside that network, some deviations from the normalized structure of bus-connection-bus nodes may occur, even after simplifying adjacent connection nodes and LV links. These situations include adjacent key nodes, and connection nodes directly joined to consumer nodes or single degree key nodes.

For example, in the source data, each distribution transformer node is connected directly to one or more fuses, which are then connected to the rest of the graph, as shown in Figure 4.10. Since both the transformer and fuse nodes are identified as key nodes, they are directly represented in the final diagram. However, to comply with the required bus-connection-bus structure, essential

for the layout and plotting logic, a bus node is necessary above the fuses and below the distribution transformer.

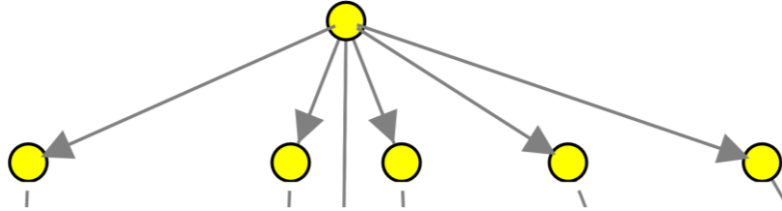
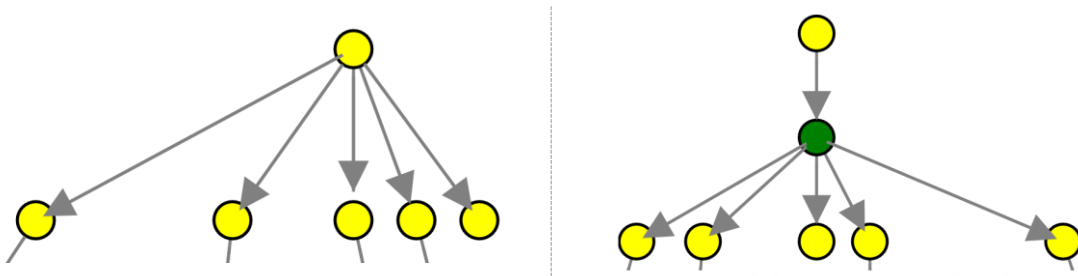


Figure 4-10: Adjacent key nodes, requiring an additional bus node (Represents a distribution transformer and its connected fuses).

To satisfy the structural requirement, additional bus nodes are created and added at the specific points in the network. Adding these nodes does not modify the basic topology of the network, as they only serve as auxiliary nodes that help achieve a normalized node structure, but the basic connectivity between every element in the network is retained.

There are two main cases that require the insertion of bus nodes. First, when key nodes are adjacent. In this case the upstream node may have several key node successors, but it is guaranteed to only have one predecessor by the convert-to-directed algorithm. Second, when connection nodes (either key or non-key) have a consumer or single degree key node successor. In this case, connection nodes will always have degree two.

In the first case, a bus node is inserted, and linked between the upstream key node, and all original successors of the *key* predecessor node are connected downstream from the bus node. For the second case, the new bus node is simply connected between the connection node and its successor.



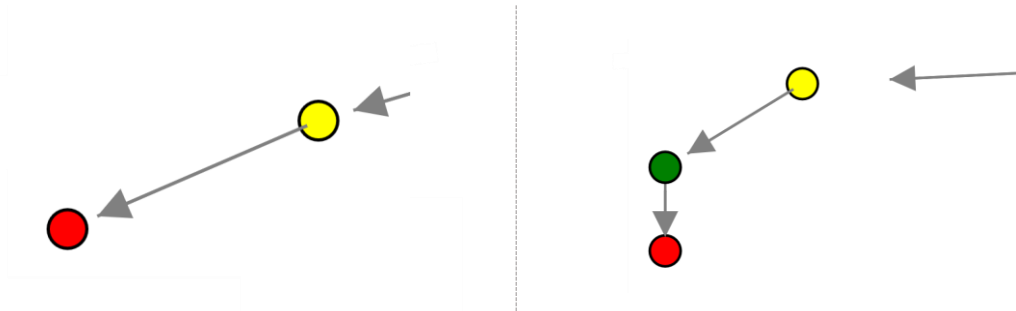


Figure 4-11: Two scenarios requiring additional bus nodes (left), and the resulting structures after adding the bus nodes (right).

Additionally, the connection nodes simplification algorithms can lead to single-degree bus nodes being created. The situation will only occur if a bus node with no successors has two adjacent connection nodes, as shown in Figure 4.12. This is also a deviation from the normalized node structure, as the purpose of bus nodes is to display that their successors are connected to electrically equivalent points (bus nodes with at least one successor), or to show that different sub sections of the network are connected to each other (bus nodes with more than one predecessor). Therefore, single-degree bus nodes are removed from the graph, as well as any non-key connection nodes that may have linked the removed bus nodes.

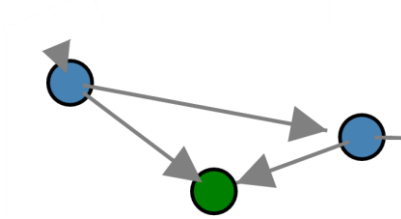


Figure 4-12: Scenario that would lead to a single-degree bus node, after processing connection nodes.

4.4.3 Graph layout generation

Inputs: Normalized graph G' , max_iterations parameter.

Output: Normalized graph G' with layers assigned, node layout.

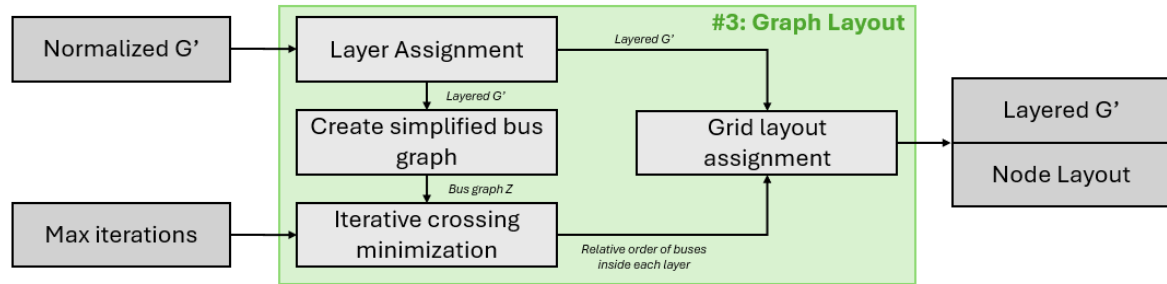


Figure 4-13: Graph layout phase.

Laying out the graph nodes is an essential step in producing a visually pleasing topological representation of the network. Generating an orthogonal single-line diagram implies considering several constraints in the placement of elements, like ensuring all connections are orthogonal. Taking as an input the normalized graph generated in the previous step, this phase involves calculating the position of each node in the graph to satisfy the following constraints:

- There are no crossings between different elements of the graph.
- The vertical spacing between different buses is constant.
- The horizontal spacing between every element in the network is constant

These requirements can be satisfied if the layout follows a grid structure, in which elements are placed in discrete x and y positions. The node's positions are therefore assigned in discrete y-layers, where the corresponding layer for each node is based on its distance to the distribution transformer *root* node(s), while the discrete grid x positions of each node are calculated to ensure the no-crossings rule is met.

Creating a layout that minimizes crossings between edges is NP hard, as discussed in the theoretical section, so exact solutions cannot be found, and heuristic methods are applied to try to find zero-crossing layouts for planar graphs. Even though all the graphs considered in this project are planar, in some cases it is not possible to generate a 0-crossing layout, because due to the fixed layer assignment process, some graphs are non-level-planar

The layout algorithm used consists in a custom implementation of the Sugiyama framework, incorporating a layer assignment algorithm, a crossing minimization optimization, and a coordinate assignment process, although with several adjustments.

Radial graphs do not require all these steps, and a simpler algorithm like Reingold-Tilford can be applied to produce a layered layout that meets the requirements. However, as it will be explained in the Case Studies chapter, almost every network that this project has considered has one or several meshed connections, requiring the custom Sugiyama Method implementation detailed in this chapter.

The layout algorithm is run on a subgraph Z that contains only the bus nodes in G' , which improves the crossing minimization performance and reduces computational load. Then, taking the relative positions of the bus nodes in each layer and the number of successors each bus has in the complete graph G' , a discrete x and y position is assigned to them. From the positions of the bus nodes, the position of every other node in the graph is then calculated (this process is further explained in the following sections).

As explained in the solution overview, the diagram generation does not render edges, as the connections are drawn implicitly through symbol placement and adjusting the sizes of buses and vertical connectors. For every bus node, the plotted bus line spans from its own position to its farthest successor, all the successors to the bus node will be placed from the bus node x position to the right. Using the coordinate for each node as the top-most reference point for drawing each diagram symbol, as explained before, allows to represent the connectivity just by the position of each symbol and the length of the bus, as shown in Figure XXX. While the symbol assignment process is detailed in the plotting section, the coordinate assignment ensures a compatible layout, by placing every bus node's successor on the same y -coordinate as itself, and on relative x positions beside the bus node, allowing the implicit symbol placement connections to be drawn.

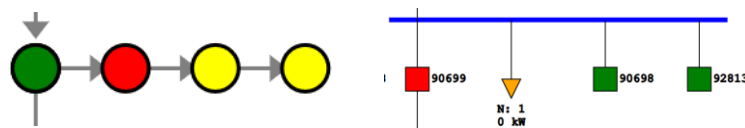


Figure 4-14: Node position assignment for elements inside the same bus.

Layer assignment

The layer assignment process is done through a maximum distance algorithm, which, starting from the root nodes, (generally corresponding to distribution transformers) traverses the graph through BFS, assigning each node its corresponding layer.

The layer assigned when doing BFS is only incremented for bus nodes, as it is desired that the successors for each bus are placed on its same y coordinate. In this way, n layers are created, where n is the longest sequence of bus nodes connected to the distribution transformer. If the algorithm finds a node whose layer is already defined when traversing the graph through a different path, the node's layer is reassigned, so that every node is placed in a layer that reflects its longest distance to every root nodes. The layer assignment logic can be observed in Algorithm 4.3.

Algorithm 4.3: ASSIGN_LAYERS

Input: Rooted normalized DAG G'

Output: Returns G' with each node assigned an integer
layer $\in \{1, 2, \dots\}$;

```

// initialise
1 foreach  $v \in V$  do
2    $v.layer \leftarrow \text{none};$ 
3   if  $\text{indeg}(v)=0$  then
4      $v.layer \leftarrow 1;$ 
5      $\text{nodes\_to\_assign} \leftarrow \text{nodes\_to\_assign} \cup v$ 
6  $\text{current\_layer} \leftarrow 1;$ 
7 while  $\text{nodes\_to\_assign} \neq \emptyset$  do
8    $\text{current\_layer} \leftarrow \text{current\_layer} + 1;$ 
9    $\text{next\_nodes} \leftarrow \emptyset;$ 
10  foreach  $v \in \text{nodes\_to\_assign}$  do
11     $v.layer \leftarrow \text{current\_layer};$ 
12    foreach  $s \in \text{successors}(v)$  do
13       $s.layer \leftarrow \text{current\_layer};$ 
14      if  $s.connection\_node$  then
15         $\text{next\_nodes} \leftarrow \text{next\_nodes} \cup \text{successors}(s)$ 
16   $\text{nodes\_to\_assign} \leftarrow \text{next\_nodes};$ 
17 return  $G$ ;

```

Creating simplified bus graph

Layout algorithms that rely on heuristics to generate a layout that minimizes crossings become less effective as the number of nodes in the graph increases and are less likely to find an optimal or near-optimal solution. Therefore, to increase the likelihood that a zero-crossings layout solution is found, it is best to provide the algorithm with a graph that contains the least possible number of nodes.

Given that the input graph to this phase is the DAG with normalized bus-connection-bus node structure discussed earlier, knowing the relative positions of the bus nodes that ensure no crossings would be enough to create a layout for the complete graph. Once the bus relative order is determined, the positions of every connection node and single degree node can be assigned deterministically based on their predecessor and successor bus nodes positions without introducing any additional intersections.

Therefore, the layout problem can be simplified from finding a position for every node, V' , in graph G' , to finding a position for n (bus) nodes, with $n \subset N$ that minimizes crossings. This both improves the likelihood that the resulting graph will have either zero or minimal crossings, and reduces the computation time required, essential because the time complexity of the layout algorithm is $O(n^2)$.

The simplified graph is therefore generated from the bus nodes in the original graph, and the edges that connect the buses with each other. However, in the original graph, the bus nodes are not neighbors directly but rather connected through connection nodes placed in between them.

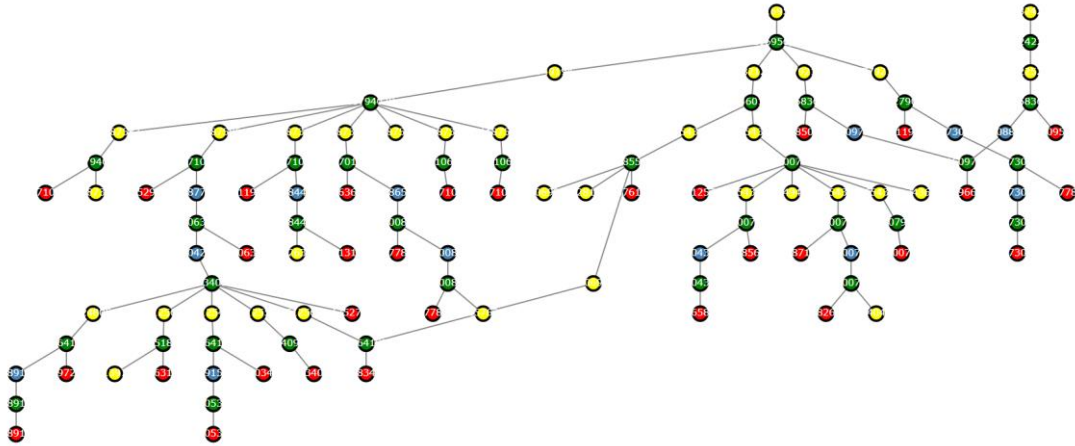


Figure 4-15: Example complete normalized graph G' .

Therefore, to generate the simplified graph, the whole graph is traversed in search for bus nodes. When a bus node is found, it is added to the simplified graph, and its successors are iterated over. The successors of each successor in the iteration are checked, and if any of them is a bus node, the corresponding node and edge is added to the simplified graph. In this way, from a complete graph like shown in Figure 4.15, the simplified graph in Figure 4.16 is created.

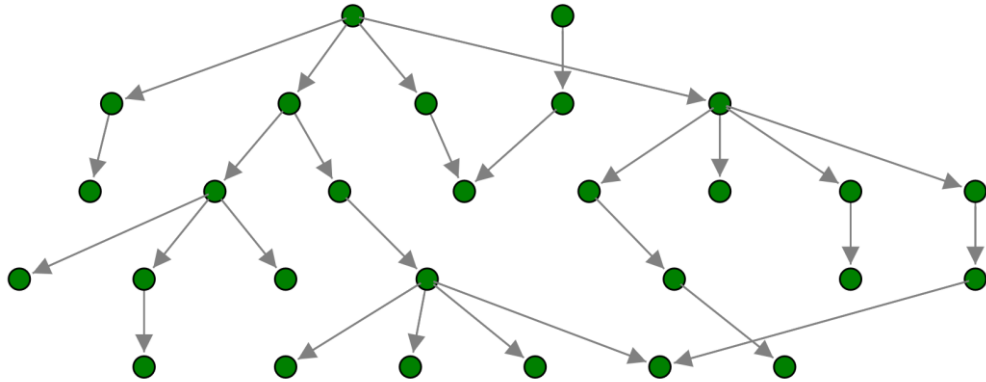


Figure 4-16: Resulting bus simplified graph corresponding to G' in Figure 4.14.

The simplified graph obtained contains the same topology as the original graph, but thanks to its reduced node count, the crossing-minimization algorithm performance is significantly improved, which corresponds to the next step.

Crossing minimization

As explained before, one of the requirements for the nodes positioning is that no crossings can be present in the final layout. To achieve this, the relative positions of the bus nodes in each layer in the simplified graph ensure no crossings have to be calculated.

The implementation of the crossing minimization algorithm is done through an external library, Graphistry, which has a *SugiyamaLayout* function that performs a heuristic barycenter algorithm to find a layout with minimal crossings, doing 2 passes over the graph.

A key requirement of this algorithm is that the layered graph only has edges spanning from one layer to the successive layer, because the algorithm does not properly minimize crossings for edges spanning more than one layer. To ensure this, dummy nodes are added to edges spanning more than one layer in the simplified graph. This dummy node feature is already implemented in some Python wrappers of the Sugiyama method, like Graphviz, however with the custom implementation around Graphistry's heuristic minimization it is necessary to add the dummy nodes beforehand.

After the dummy nodes are added, the crossing minimization layout algorithm is run. A single run of the algorithm does not produce optimal results, because the heuristic approach followed by the algorithm, consisting of assigning each node the average position of its neighbors over two passes, is very prone to getting stuck in local minima, which are not suitable as it is necessary to find a zero-crossings solution for the final diagram to be valid.

To improve the likelihood of the algorithm to arrive at an optimal (zero crossings) solution, the algorithm is run iteratively (maximum set to 100 runs), and after each run, the number of crossings of the generated layout is calculated through a helper function. For every run, the order of the nodes input into the algorithm is randomized, to force the heuristic algorithm explore paths that might lead to an optimal solution and not get stuck in local minima. If a solution with 0 crossings is achieved, the algorithm stops and that layout is taken as the optimal solution. If it is not found, the solution with the least number of crossings is returned after 100 runs.

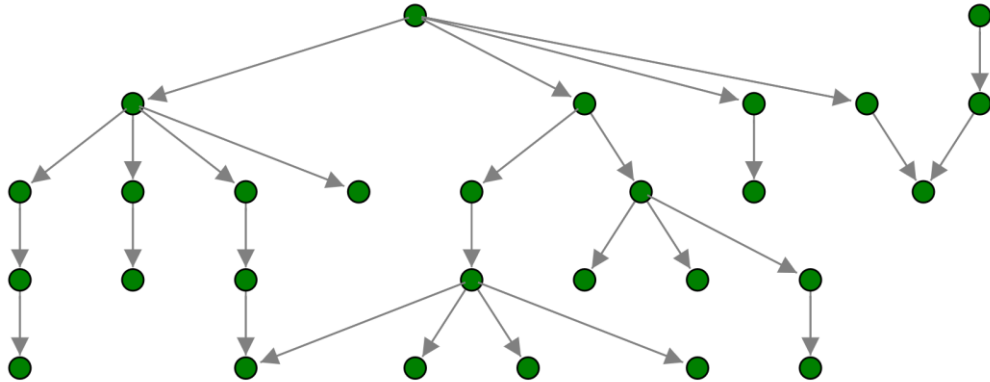


Figure 4-17: Zero crossings layout of the graph in Figure 4.16.

As the only change between iterations is randomly changing the order of the input nodes, this is a Naïve algorithm. Due to the nature of the problem, it has not been possible to find a deterministic solution, as randomness is the only input that can lead the algorithm to explore the whole solution space and find an optimal solution (if one exists), while near-optimal solutions provide no value. In practice the algorithm is able to deliver optimal layouts for medium size graphs (the ones analyzed are around 20-200 nodes), in less than 10 iterations, however this is further explored in the case studies and results chapters.

The algorithm however has a significant limitation, which is non-level-planar graphs. As the layer assignment is fixed and based on distance from the root node, some graphs might have a layer assignment incompatible with a zero-crossings layout, making it impossible to find an optimal solution for the crossing minimization algorithm.

The output of this algorithm is the relative position of the nodes in each layer in this layout is used to calculate the grid positions for the bus nodes in the graph, which is then used as the input for calculating the positions of the nodes in the whole graph.

Generating the bus graph layout

The result from the Sugiyama layout provides the relative position of each bus in the graph to minimize crossings. However, the absolute positioning of each node does not meet the objective of adjusting the nodes into discrete x positions for plotting it into an orthogonal diagram.

The discrete y layers are already calculated, and the nodes on each layer are known. Therefore, it is necessary to calculate the x positions of the bus nodes that ensure nodes vertically placed

in discrete columns, to allow every node in the final layout to be orthogonally connected to its neighbors. For this purpose, this function assigns grid positions to each of the buses in the simplified graph, taking the same relative ordering of nodes per layer as given by the Sugiyama layout algorithm.

As mentioned in the introduction to this phase, successor nodes of the same bus node are placed on the y position of the bus node, and on discrete x positions beside it, to its right. Therefore, the positions of the buses to be laid out must be such so that later, when laying out the rest of the nodes in the graph, there is enough space to place each bus node's successors to the right of them, and no intersections are caused by this node placement. To achieve this, a property called *size* is added to the bus nodes of the simplified graph Z, which is equal to the number of successors in the total graph. The final assigned x positions of the nodes ensure that every bus node has at least *size* available x discrete positions to its right.

The x assignment algorithm uses a queue of nodes, to which new nodes are dynamically added and allows to adjust the x position of each node in order. First, the leftmost node with no predecessor is taken, which is assigned an x of 1, and the result of its $size + x$ position is assigned to the corresponding layer's x restriction function. Then its successors are added to the queue, starting with the leftmost successor (according to their laid out position), which is assigned the same x value, and gets its $size + x$ position restriction recorded into the dictionary using the `addx` position function, its successor(s) are then also recorded into the queue, ensuring that the node to be processed next is the leftmost of its successors. This process is repeated iteratively with the $x=1$, until a node with no successors is found. This means a new column has to be created, and its x size is obtained from the dictionary of min x positions, taking the value corresponding to that layer.

To achieve this, the function requires a helper function, which stores the minimum x position restriction to add the following columns in, for each layer. It ensures that new columns of nodes do not intersect with lower layers, by when adding a new restriction on a given layer y, overwriting lower restrictions on upper layers until a higher restriction is arrived at.

When new sizes are assigned to the dictionary of x restrictions, the `add x` restriction function propagates the value upwards if it is higher than the restrictions above, ensuring that when placing a new column, the downstream nodes do not cross already placed nodes.

An special case occurs when a node with two predecessors is reached (three predecessors or more case is not contemplated, as it is ensured it cannot happen in previous functions). If that node does not have an x position assigned (i.e., it is the first time it is visited), its x position is assigned normally, as part of the current column). If it does have an x position assigned, its position is not changed, and two cases can occur:

1. The x difference of the predecessors nodes positions is lower than the node's "size" property, therefore when the successor nodes are added to the graph, they would not fit beside the bus node. To solve this, the x position of the whole column is displaced by the difference in size of the bus and the gap between the positions of the two predecessors.
2. The x difference of the predecessors is enough to fit the size of the bus, in which case the column would not need to be displaced. However, if the rightmost predecessor has other successors, a gap is left below the predecessor before laying out the rest of its successors, to ensure the corresponding connection node to the two-predecessors node can be fit (this also could apply to the first case).

To solve the first case scenario, a helper function is required that returns the topmost element in the current column, so that it can be added back to the queue of elements to be assigned with the new restriction in place. This function is a recursive call that checks for predecessors if they are in the same column, calling itself if it finds a predecessor in the same x position. The recursive call stops when a node with a predecessor with a different x position is found, and that node is returned. The node's id is then assigned to the first position of the queue, and an x restriction is added to its layer equal to the x position of the two-predecessor-node + its size. This way, the x positions of the nodes in the column are re-calculated and when the algorithms reaches the 2-predecessor node again, the first case scenario is not triggered anymore.

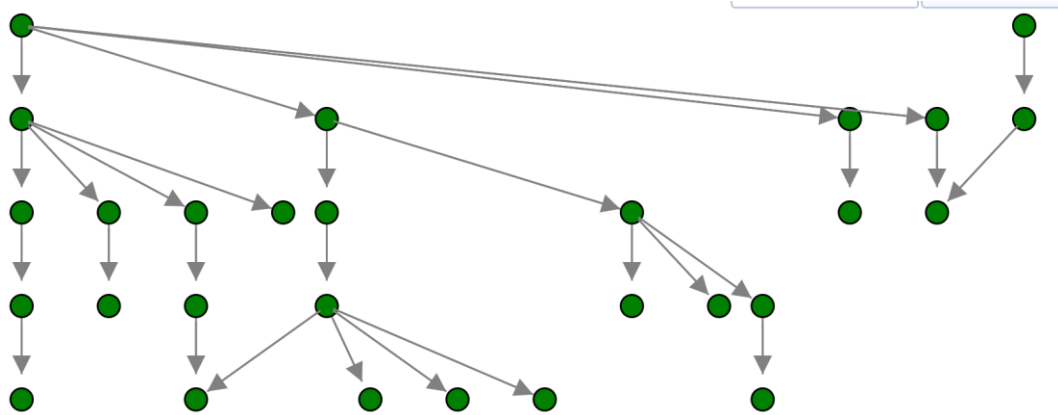


Figure 4-18: Grid layout of the simplified bus graph in Figure 4.15.

Once all nodes have been processed, which is detected by the queue being empty, the algorithm returns the resulting position dictionary for the bus nodes in the simplified graph.

Generating the complete graph layout

Taking as an input the positions of each bus node in the simplified graph, and the complete graph, the function reconstructs the whole graph by calculating the positions of every node from the positions of the buses.

The first step is the assignment of connection nodes positions. In general, connection nodes are placed vertically on the same column as their successor bus node, and in the same row as the predecessor bus. However, there is an exception to this case, for nodes that have two predecessors, the preceding connection node that is on the higher x position cannot be placed directly on top of its corresponding feeder node because the two nodes would overlap with each other and, potentially creating new intersections that invalidate the final diagram. This special case is already considered when placing the bus nodes, as a vertical column gap is put in place to allow placing the connection nodes connecting to buses on a lower x position directly on the same column as their predecessor nodes.

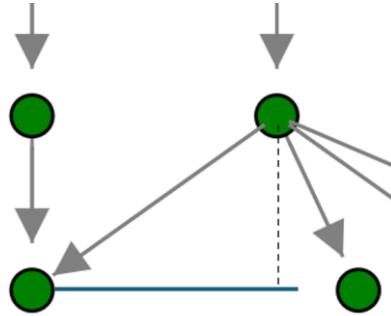


Figure 4-19: Column gap to allow the placement of bus nodes with two predecessors, in the layout in Figure 4.18.

Therefore, to calculate the x position for every connection node in the graph, the x position of their preceding bus node and their successor bus node are compared. If the x position of the predecessor node is less than or equal to the x position of the predecessor node, the x position of the connection node is set equal to the successor's x position. However, if the x position of the successor node is lower (which can only occur for nodes with two successors), the connection node is placed on the same vertical column as its predecessor.

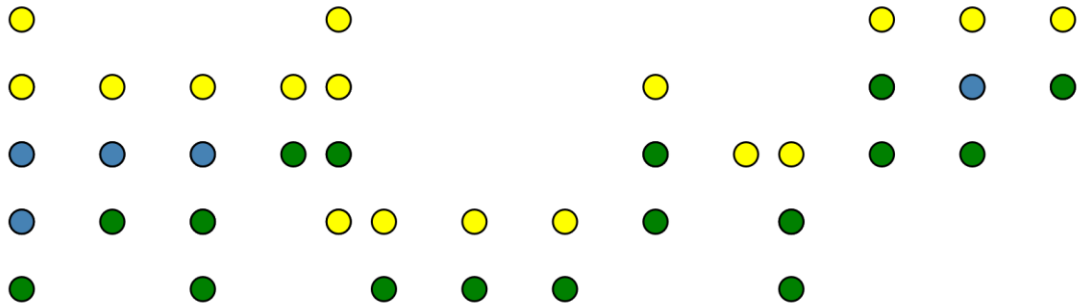


Figure 4-20: Connection node positions.

There is an additional scenario to consider for nodes with two successors, the existence of dummy nodes in the simplified graph that constraint the x position of the connection node. As these nodes are not present in the complete graph, when checking the predecessor and successor of a given connection node, the algorithm will fail to consider the x position of any dummy nodes in between if the x position of the real bus preceding the dummy node(s) is different to them. In this situation, the predecessor's x position may be incorrectly used to calculate the x position of the connection node.

To solve this situation, it is necessary to check whether each connection node is preceding a bus node with more than one predecessor and with either of the predecessors being a dummy node (only situation in which a bus node will have dummy nodes preceding it is if the bus has more than one predecessor, with each predecessor having different hierarchies). The algorithm then calculates the first non-dummy bus predecessor of the dummy node chain through a helper function that recursively searches through the predecessors of each dummy node (dummy nodes can exist standalone or as a chain of n nodes for a difference n in hierarchy between connected buses). When the non-dummy predecessor is found, the algorithm compares the non-dummy predecessor to the predecessor of the connection node being checked. If it is the same node, it means that the dummy chain corresponds to that connection node, and therefore, the connection node's position might be constrained by the dummy chain; therefore, the x predecessor position used for the connection node x calculation logic should be the first dummy node's x position. If it is not the same node, the dummy chain does not correspond to that connection node, so the predecessor's x is calculated as normally.

After all the connection nodes are laid out, the remaining nodes have to be placed on their corresponding grid positions. Remaining nodes include degree 1 nodes like demand and key nodes (including open disconnected LV links, for example), and root nodes (distribution transformers or primary substations).

Degree 1 nodes are assigned to their positions based on their preceding feeder nodes positions. First, feeder nodes are searched for, and the successors of each are iterated over. If the successor is a connection node it is skipped, as its position is already assigned. If it is not an connection node, an iterative search of an empty x position in its corresponding layer in the grid to the right relative to the preceding bus node's x position is conducted. An empty x position is found if there are no connection nodes or other degree 1 nodes in that position, the bus node's x position is considered empty too if there are no connection or degree 1 nodes already placed in that position, as explained previously.

As the minimum size of each bus node is considered when calculating the positions of the bus nodes in the grid, it is guaranteed that for every node in the grid an empty x position is available that does not cause intersections with other buses. If this is not the case it would mean that the size of the buses has been incorrectly calculated.

The last step is calculating the position of the root nodes. As these nodes have no predecessors their position has is not calculated in the previous steps of the algorithm. The positions of these root nodes are less restricted than other nodes in the graph, as long as they are placed over any point spanning the x length of their successor bus they will cause no crossings with other sections of the network. They could be placed directly on top of the bus node for simplicity, but to obtain a more aesthetically pleasing final diagram they were placed on the x column corresponding to the middle point of the successor bus, which is calculated as the middle point between the bus itself x position and the bus's farthest successor node's x. The root nodes y position is calculated based on its hierarchy level too, which in this case corresponds to the first layer in the diagram.

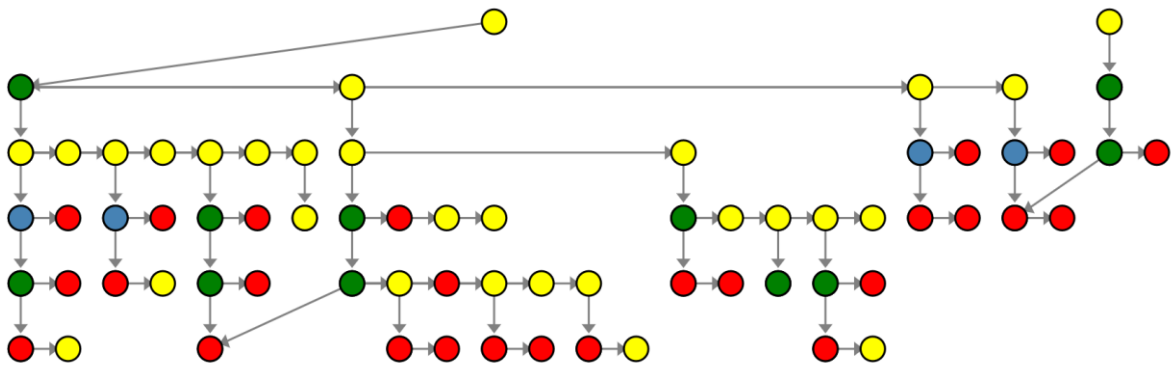


Figure 4-21: Final complete graph layout, for the sample graph in Figure 4.15.

4.4.4 Schematic plotting

Generating the diagram plot takes as the main input each node's properties and coordinates calculated previously. The coordinates for each node represent the topmost (and leftmost for buses) reference point to draw each of the diagram symbols. The graph's edge information is not required by the routine, as in the layout process the nodes are assigned positions which ensure that by plotting a symbol to each node and adjusting the bus and connection nodes length, the diagram generated reflects the connectivity rules of the underlying graph and, therefore, the real network.

Calculating length

An additional property that must be calculated however is the length for the buses and vertical connectors. As the edge data is not used to generate the diagram, this is a necessary variable because the connections are drawn implicitly, and the position assignment is calculated so that buses and connections graphically span over different distances, generating the graphical connections.

Buses, which are represented by horizontal lines, graphically have a length that spans from the x position of its first to last successor, or the x position of its predecessors (in case it has more than one), whichever is higher. Nodes with a single predecessor and successor are therefore assigned a length value of zero. Connection nodes, which are represented as vertical lines, are assigned a length equal to their own y position (which is the same as their predecessor bus node's y position) minus the y position of their successor. Therefore, the minimum length assigned to any connection node is 1 (nodes connecting adjacent layers). Figure 4.22 (same as 4.13) shows an example bus node, which has four adjacent successors. The bus node is therefore assigned length = 3, which therefore implicitly generates the connectivity information graphically.

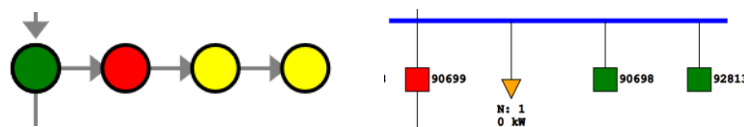


Figure 4-22: diagram plot of a length=3 bus node (same Figure as 4.14).

Generating data structure output file

Once node sizes, positions, and other node properties that might be relevant to include in the diagram are determined, they are exported into a CSV file. This CSV file serves as an universal input for any plotting routine to generate a visual representation of the graph in any desirable format.

For this project, a plot to SVG script has been created, which reads the csv and assigns a custom diagram symbol to each node, generating a self-contained output vector file that contains the diagram visualization in SVG format. Some of the diagram symbols created for the most common nodes found in the data can be observed in Figure 4.23, along with some additional properties displayed.

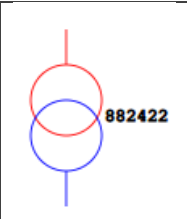
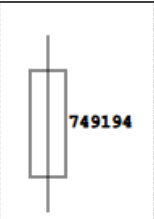
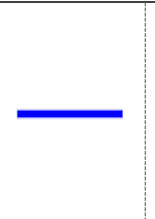
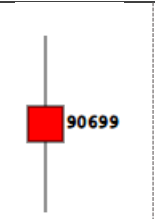
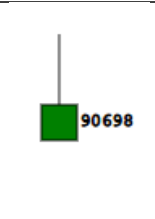
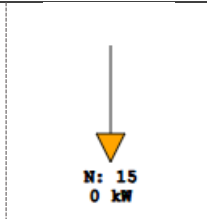
						
Node type	Distribution Transformer	Fuse	Bus	Closed LV Link	Opened LV Link	Consumer group
Properties displayed	· Node ID	· Node ID	· None	· Node ID · Status (color)	· Node ID · Status (color)	· Number of consumers · Total demand

Figure 4-23: Schematic symbols used in the final diagram.

The SVG (Scalable Vector Graphics) format was chosen to generate the plot due to its versatility, scalability, and broad compatibility across platforms and devices. Representing the diagram in a vector format is optimal due to the geometric structure of the underlying data. This approach allows more efficient storage and manipulation compared to raster image formats like PNG or JPEG, which not only would require significantly more space, but also would not preserve resolution when scaled.

The resulting diagram plot from the example graph used in the layout section can be observed in Figure 4.24.

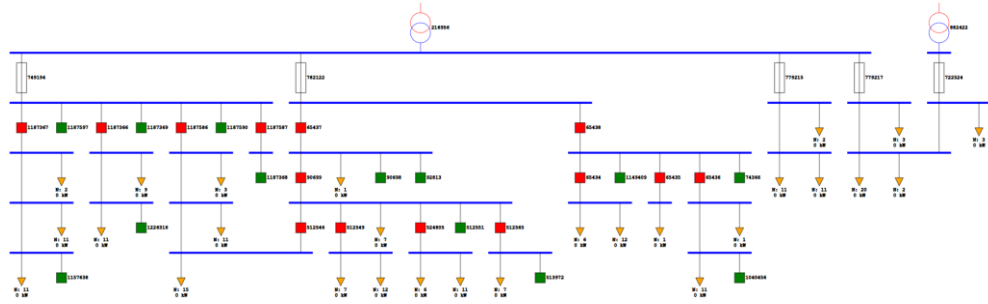


Figure 4-24: Final diagram, for the sample graph in Figure 4.15.

5 Case Studies

To validate the results of the automatic diagram generation algorithm, a case study approach is used, in which several individual samples of LV networks of a wide variety in size, and structure, are represented as diagrams using the approach described in the methodology section, measuring several metrics to gauge the performance of the algorithm and determine if the produced diagram is valid.

5.1 Experimental Setup

5.1.1 Overview of network characteristics

Eight case studies are conducted on networks spanning a range of sizes and topological complexities: small (<500 nodes), medium (500-2000 nodes) and large (>2000 nodes) sized networks. The latter case studies feature the most demanding networks. Key characteristics of the 8 networks can be observed in Table 5.1. Topological complexity is quantified by the number of meshed connections (M), which is equal to the number of cycles in the graph, or equivalently, the number of bus nodes with two predecessors in the final graph. A higher value of meshed connections increases the difficulty of producing a crossing free layout and allows to test the performance of the layout algorithm on increasingly complex networks.

Table 5-1: Case study networks characteristics

Case Study	NNODES	M	NCONSUMERS
#1 Small Radial Network	69	0	14
#2 Small Quasi-Radial Network	145	1	17
#3 Small Meshed Network	239	2	71

#4 Medium Sized Network 1	611	2	208
#5 Medium Sized Network 2	1126	5	482
#6 Medium Sized Network 3	1233	4	472
#7 Large Network	2282	11	844
#8 MV Network	1810	7	0

5.1.2 Result validation

Each case study will measure several metrics in addition to validating the final diagram, to compare the performance of the algorithm under different scenarios. The metrics that will be measured include:

- k: number of crossings in the final layout
- N_iter: number of iterations of the layout algorithm required to achieve that number of crossings
- N_L: number of nodes laid out in the final graph
- R: runtime in seconds (s).

The resulting diagrams will only be considered valid if the value of k is equal to 0, as a layout with crossings will cause elements in the final diagram to graphically overlap with each other, generating diagrams that represent incorrect connectivity information.

5.1.3 Input parameters

To generate comparable results among different case studies a set of standard input parameters is used, which can be observed below:

Max_consumers_per_group = 100
Max_consumers_per_bus = 100
Max_layout_iterations = 100

These parameters ensure that only the key topological information is reflected in the final diagram, which is desirable to compare the results among different case studies. The effect of increasing or decreasing the max consumers variables is validated in case study 1.

The value of max layout iterations serves as a baseline for testing whether the algorithm is able to find an optimal solution in a reasonable number of iterations. The effect of further increasing this metric for some graphs will be analyzed in the results section of this chapter.

5.1.4 Objectives

Each case study has a slightly different focus; the first three case studies examine how the classification and simplification algorithms process the underlying network data to generate an accurate representation of the topological characteristics of the network. On the other hand, the next five case studies focus on analyzing the performance of the layout algorithm and testing its ability to generate crossing free results, and therefore valid diagrams. The last case study additionally tests the versatility of the algorithm by generating a MV network diagram, verifying its capacity to use data with different characteristics.

5.1.5 Hardware:

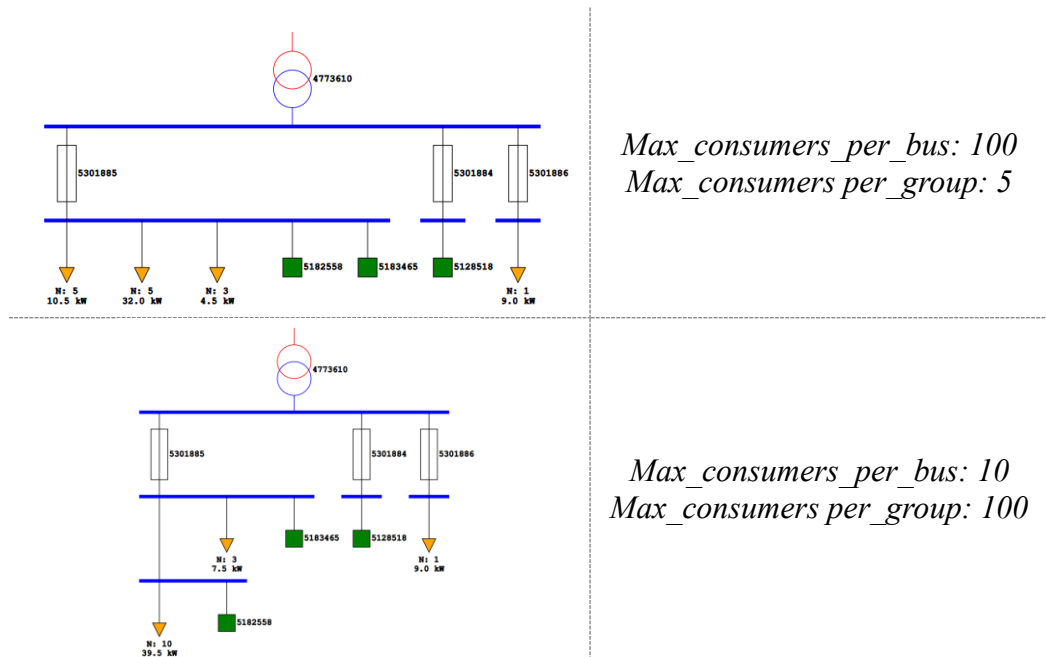
The case studies were performed on a standard laptop with 13th Gen Intel Core i7-13620H and 16 GB DDR4 RAM. All timings reflect single-threaded execution.

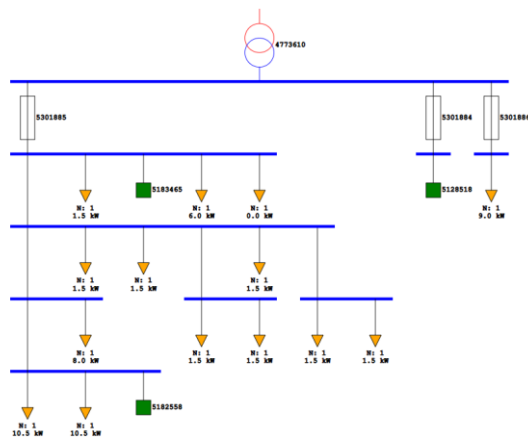
Total Nodes Laid Out: 13
Meshed Node Count: 0
Runtime: 0.102683048248291 seconds

Conclusions of the case study

The algorithm produced a diagram with the expected result given the standard input parameters, as it grouped all consumers along each one of the feeders, along with the corresponding opened LV links in each feeder. The fuses connected to each of the transformer's feeders were also represented, each with their corresponding id.

The values for *max_consumers_per_group* and *max_consumers_per_bus* were set to the standard values explained before, 100. However, to validate the impact of these parameters on the final diagram, three additional diagrams have been generated, each with a different set of parameters. The results can be observed in Figure 5.3:





Max_consumers_per_bus: 1
Max_consumers_per_group: 1

Figure 5-3: Case study 1 parameter adjustment results.

The results prove that decreasing the value of *max_consumers_per_group*, while holding *max_consumers_per_bus* constant, decreases the sizes of consumer groups on each bus, and therefore a larger number of groups is required to represent all consumers, adjusting the level of horizontal detail in each bus. Conversely, decreasing the value of *max_consumers_per_bus* increases the level of detail into the network's vertical structure, as it shows more intermediate buses and therefore displays the electrical distance between different elements in the network more exactly

In the third diagram, it is demonstrated that setting *max_consumers_per_bus* to 1 does not guarantee that the final number of consumers per bus is going to be exactly 1. As explained in the methodology section, if several consumers are connected to the same node in the graph, they are at the same electrical distance and therefore all appear on the same bus in the final diagram. Additionally, some simplification steps can lead to this situation happening even if they were not exactly on the same node, but at equivalent distances.

Overall, the results to this case study have proven the algorithm produces the expected results in small radial networks, and that the parameters can be used to tune the final diagram, but some properties are not controllable

5.2.2 Case Study 2: Small Quasi-Radial Network

The second case study takes as an input a small network, with almost radial characteristics. In this case, however, the network has two transformers, which due to the two root nodes causes a node to have two predecessors and therefore, the graph has a meshed connection. The objective of this case study is to test the algorithm on an additional simple network, to which the resulting schematic can be intuitively compared, and to test if it can successfully handle the meshed connection to ensure the final layout has no crossings.

This network section is located in south Glasgow, in the Kingston neighborhood. It features 17 consumers, 2 transformers and a single opened LV Link. One of the transformers is only connected through a single feeder to the rest of the network, while the other has 5 feeders that branch out radially to the 17 consumers. Its GIS representation can be observed in Figure 5.4:

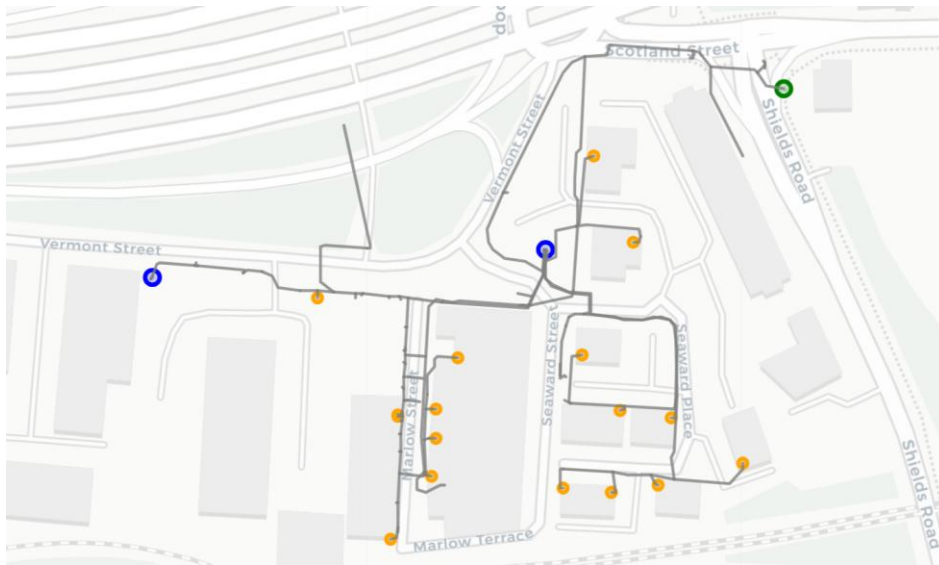


Figure 5-4: Case Study 2 network GIS representation.

Output

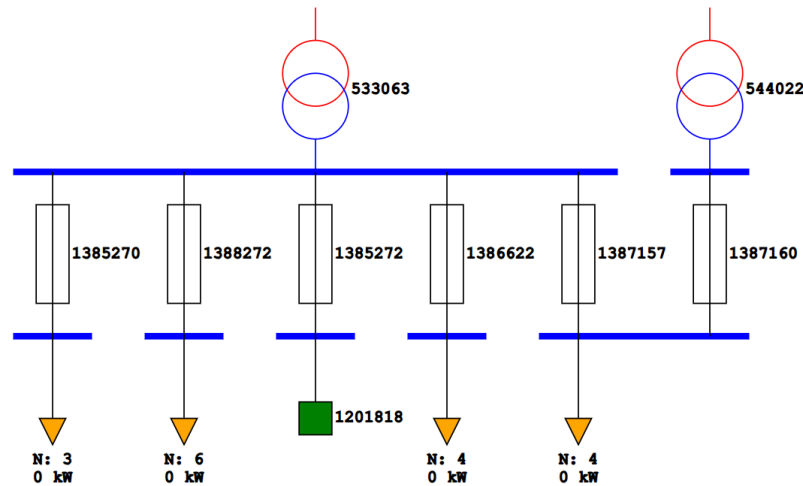


Figure 5-5: Case study 2 output diagram.

Graph laid out with 0 crossings, achieved after 1 iterations
Total iterations: 1
Total Node Count: 145
Consumer Node Count: 17
Total Nodes Laid Out: 15
Meshed Node Count: 1
Runtime: 0.12869524955749512 seconds

Conclusions:

The resulting diagram aligns with the expected result. The consumer nodes along each of the transformer's feeders, and the opened LV link were grouped and displayed downstream from their respective fuse. The two transformers are connected through a bus in the second layer, right after the fuses.

The position for the bus connecting the two transformers was also correctly laid out, as if it had been assigned a position between two feeders of the 5-feeder transformer, a collision would have been caused. This solution was found in the first iteration, meaning that a single pass of the heuristic algorithm was enough to find the optimal solution. Time to compute was slightly longer than for the first case study, which is expected given the larger size of the graph.

This case study also demonstrates that the plotting routine can correctly calculate the length of buses that have two predecessors and plot the corresponding bus accordingly.

5.2.3 Case Study 3: Small Meshed Network

The network chosen for this case study is the larger network from which the subgraph used in the first case study was taken. Although the network's total node count is reduced, it features several meshed connections and LV links that allows to test how the algorithm handles these cases.

The network contains a single distribution transformer, several opened and closed link boxes, and 71 consumers. Figure 5.5 shows its GIS view

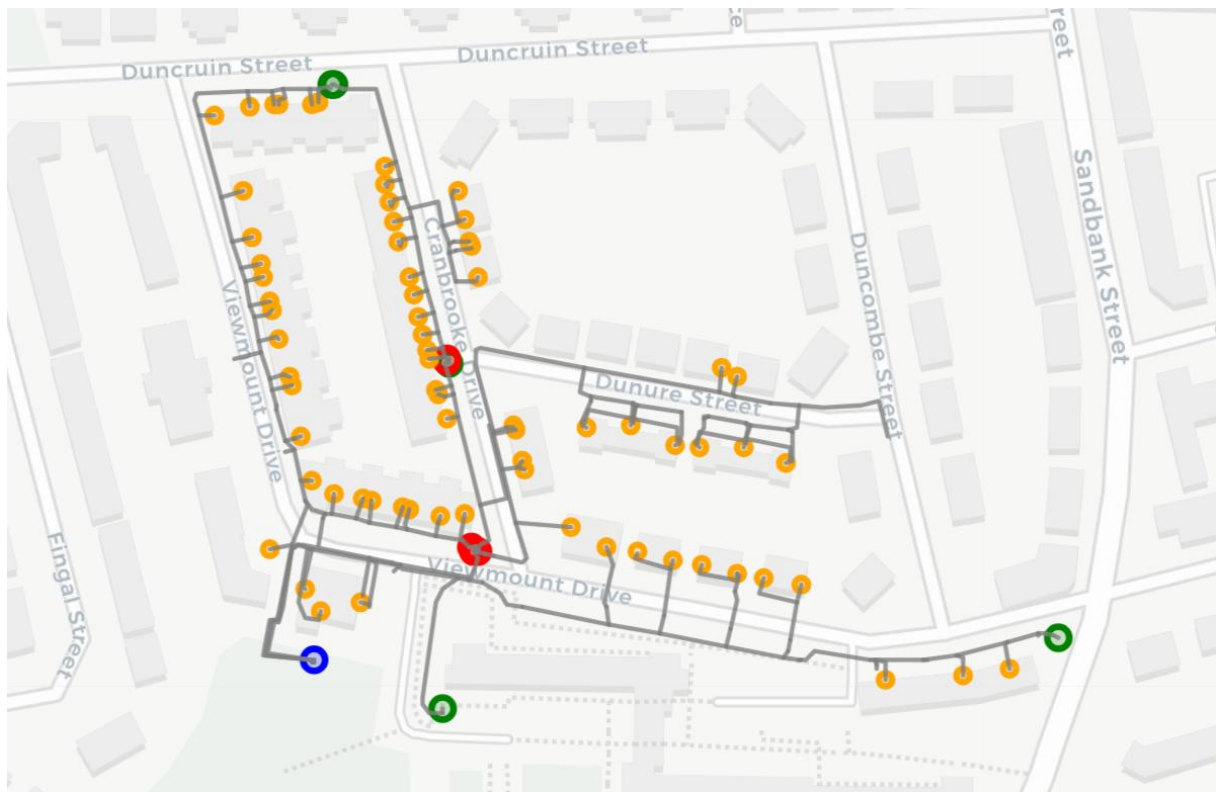


Figure 5-6: Case study 3 network GIS representation.

One particular feature of the network is that it contains both ends of an open link box (can be observed at the top of Figure 5.6), which are topologically connected to different sections of the equivalent network graph, but that the algorithm should identify by their adjacent IDs.

Output:

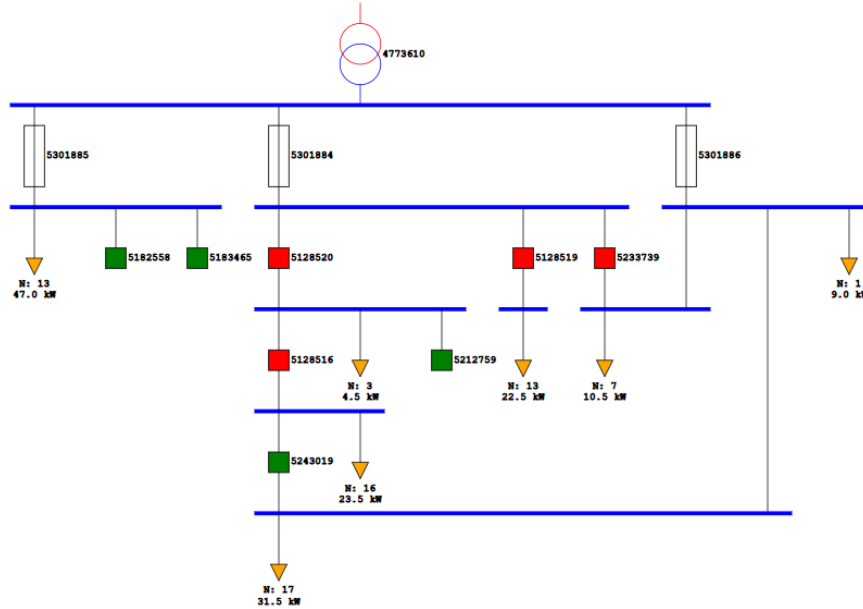


Figure 5-7: Case study 3 output diagram.

Graph laid out with 0 crossings, achieved after 1 iterations
Total iterations: 1
Total Node Count: 145
Consumer Node Count: 17
Total Nodes Laid Out: 15
Meshed Node Count: 1
Runtime: 0.12869524955749512 seconds

Conclusions:

The resulting diagram reflects the anticipated meshed topology; in this case, there are two buses with two predecessors, while the rest of the network remains radial. The crossing minimization problem is more complex for the present case compared to case study 2, as the relative position of buses over several different layers was constrained to ensure the rightmost buses connection causes no crossings. Still, the heuristic arrived at the optimal solution over a single iteration, demonstrating its efficiency and effectiveness for graphs with a low number of nodes and meshed connections.

Additionally, the algorithm correctly identified the adjacent but disconnected LV links, representing them with the standard LV link symbol (5243019), and its corresponding open state (green), while visually indicating its relationship with the two adjacent buses.

To validate the diagram's topology in more detail, each section is individually analyzed and compared to the original GIS representation in Figure 5.8.

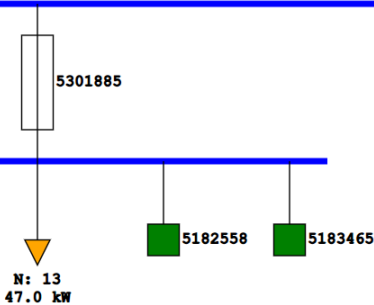

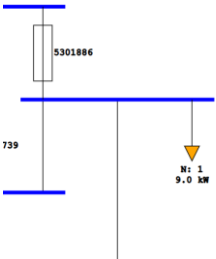
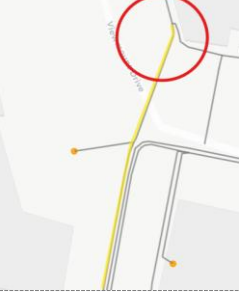
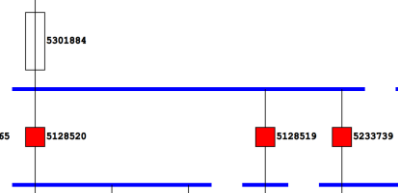
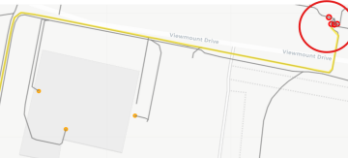
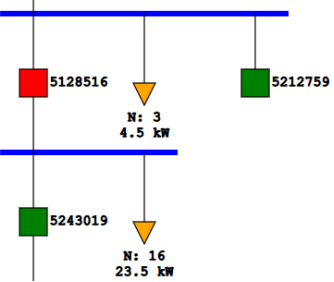

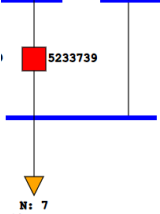

		<p>Same feeder as in case study 1, features 2 open LV links and 13 consumers (3 are not visible in the figure) downstream from the fuse</p>
		<p>Second feeder, has a single consumer connected downstream from the fuse, before branching out onto two different buses</p>
		<p>Third feeder, directly connecting to three other buses through closed LV links</p>
		<p>Subsequent buses to the third feeder, featuring 19 consumers, 16 of which are connected downstream from another closed LV Link. The open LV link, 5243019, is connected to the downstream bus</p>
		<p>Two-predecessor bus connecting one of the branches from the second feeder and the third feeder through a closed LV link</p>



Figure 5-8: Comparison of original network sections and their diagram representation.

5.2.4 Case Study 4: Medium Sized Network #1

The network selected for this case study is a 611-node section of the East Glasgow LV grid, which contains 208 consumers, 2 transformers and several LV links. Its GIS representation can be observed in Figure 5.9.

This case study aims to test the performance of the algorithm when the scale of the networks represented is increased, but their topological complexity is kept constant, as the number of meshed nodes in this network is the same as in the previous case study (2).

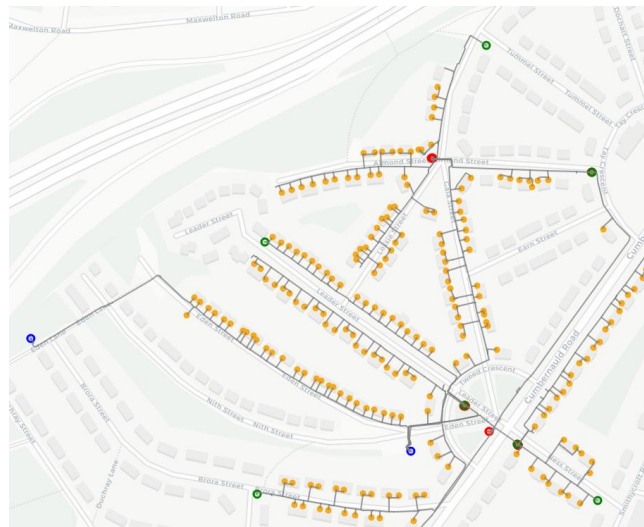


Figure 5-9: Case study 4 network GIS representation.

Output:

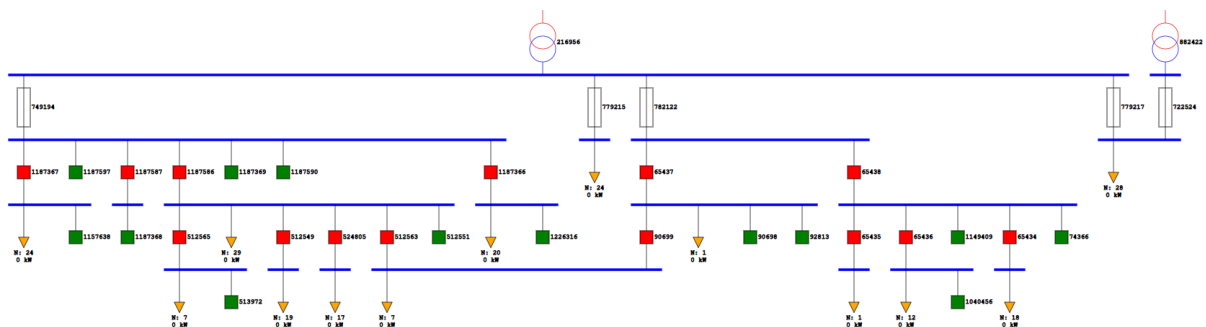


Figure 5-10: Case study 4 output diagram.

Graph laid out with 0 crossings, achieved after 1 iterations
Total iterations: 1
Total Node Count: 145

Consumer Node Count: 17
Total Nodes Laid Out: 15
Meshed Node Count: 1
Runtime: 0.12869524955749512 seconds

Conclusions:

The algorithm successfully generated the diagram without crossings, only requiring one heuristic pass of the layout algorithm., and the runtime increased linearly with the number of nodes.

This linear relation exhibits that when the number of meshed connections is constant, the algorithm exhibits the expected $O(n)$ time complexity, as theorized on the methodology section. If the layout had required more iterations, its time complexity would have likely increased, which is further tested in the next case studies.

These results confirm that the node count does not significantly affect the effectiveness of the layout algorithm, for a fixed number of meshed connections.

5.2.5 Case Study 5: Medium Sized Network #2

A larger, and more meshed network is used for this case study, which contains 1126 nodes, 482 consumers and 5 transformers. It corresponds to a grid section located in Greasby, in the Wirral Peninsula, and its geographical representation can be observed in Figure 5.11.



Figure 5-11: Case study 5 network GIS representation.

The additional meshed connections created by the increased number of transformers and redundant supply points, typical of suburban distribution grids, significantly increase the network's topological complexity. The case study therefore serves as a benchmark for testing the effect of this increased complexity and comparing its results with similarly sized networks with less meshed connections.

Output

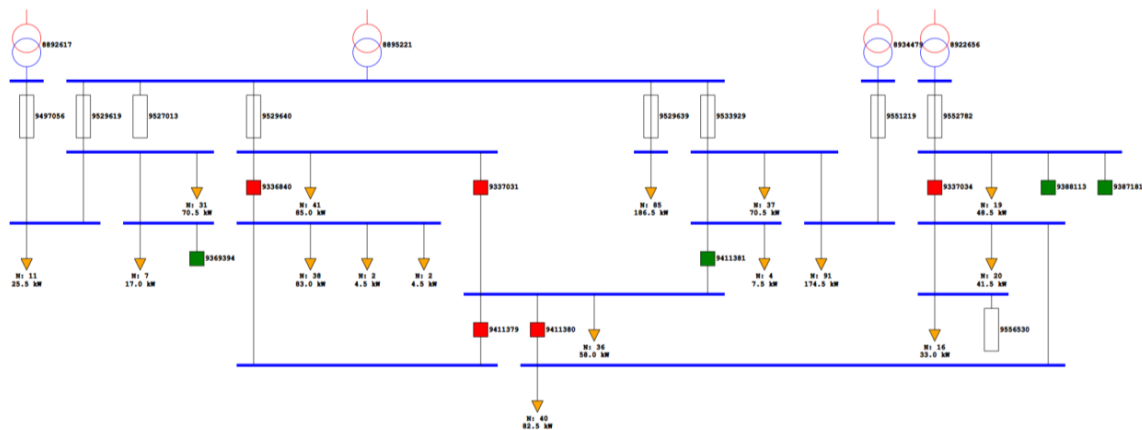


Figure 5-12: Case study 5 output diagram.

Graph laid out with 0 crossings, achieved after 4 iterations
Number of iterations 4
Total Node Count: 1126

Consumer Node Count: 482
Total Nodes Laid Out: 64
Meshed Node Count: 5
Runtime: 5.069687604904175 seconds

Conclusions

The algorithm was successful in finding a zero-crossing layout, which was achieved after 4 iterations. This aligns with the expectations due to the increased topological complexity.

The requirement for 4 iterations suggests that the first three attempts encountered local minimums in the solution space. The success on the fourth iteration demonstrates the value provided by the naïve randomization algorithm in succeeding to find the optimal solution, escaping previous suboptimal configurations.

The algorithm's runtime was 3.25 times higher compared to Case Study 2, even though its total node count was only 2 times larger, and the number of nodes laid out was smaller. This confirms that for layouts that require several passes of the heuristic layout algorithm, time complexity is not linear to the number of nodes and is most likely also driven by additional factors like the number of meshed nodes and nodes laid out.

5.2.6 Case Study 6: Medium Sized Network #3

Located in Moreton, Wirral Peninsula, the network for this case study is comprised of 1233 nodes and serves 472 consumers through 4 distribution transformers. The characteristics of the network are similar to the one used in case study 5, with several meshed connections.

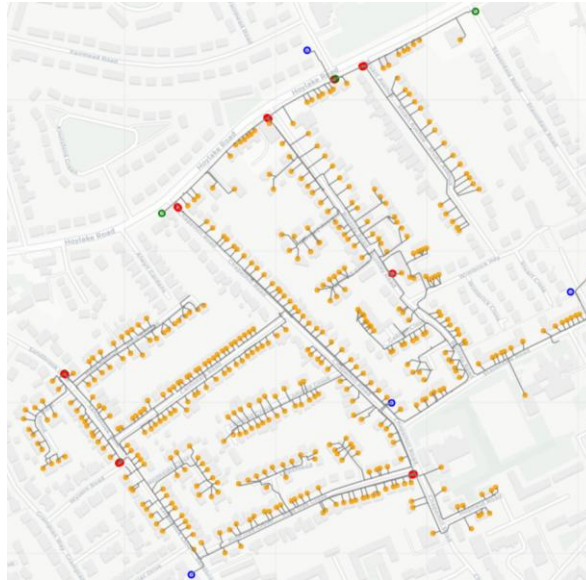


Figure 5-13: Case study 6 network GIS representation.

Output

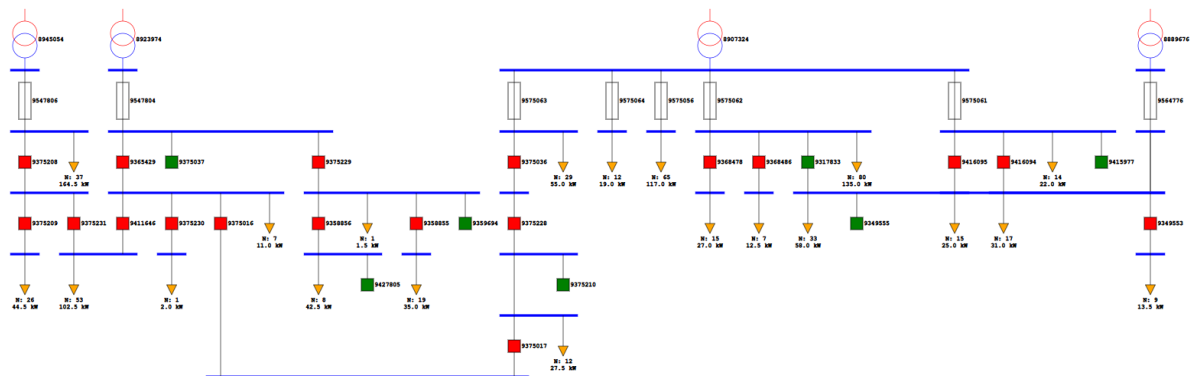


Figure 5-14: Case study 6 output diagram.

Graph laid out with 2 crossings, achieved after 20 iterations
 Number of iterations 100
 Total Node Count: 1233
 Consumer Node Count: 472
 Total Nodes Laid Out: 90
 Meshed Node Count: 4

Runtime: 6.428042888641357 seconds

Conclusions

After 100 iterations, the best solution the algorithm could find is a layout with two crossings, which was identified in iteration 20, after which it kept searching for a zero-crossings solution although unsuccessfully, producing a flawed diagram which contains overlapping elements, as shown in Figure 5.15. The diagram contains a bus with 4 predecessors, but in reality, it represents 2 overlapping buses with 2 predecessors each, which is the maximum number each bus can have. This incorrectly reflects connectivity information, making the diagram unsuitable for operational use.

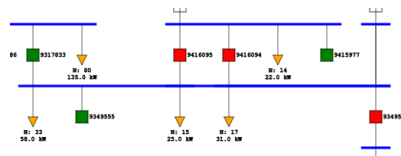


Figure 5-15: Overlapping buses in the final diagram.

The reason why the algorithm was unable to find a solution with zero crossings lies on the underlying network configuration. Figure 5.16 contains a representation of the underlying simplified graph with the layout generated by the algorithm: the two crossings are created by the relative positions of the bus nodes at the right end of the graph.

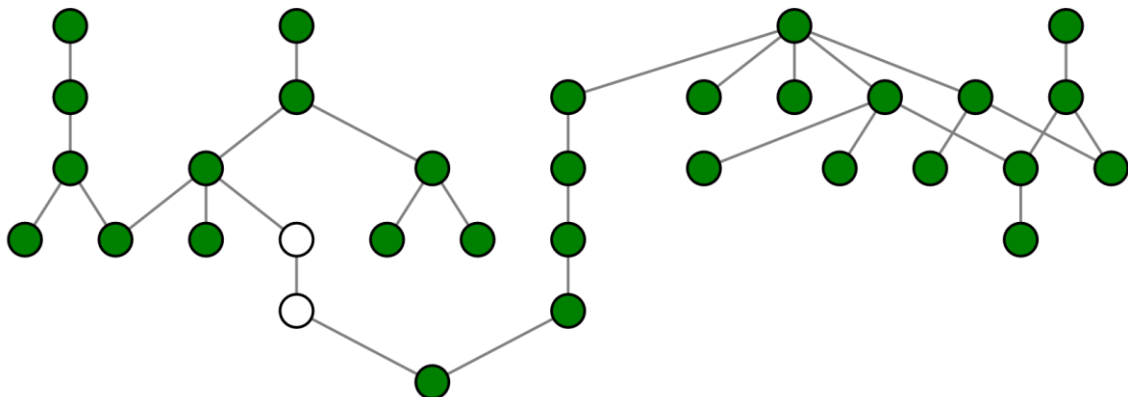


Figure 5-16: underlying non-level-planar graph of network 6.

This layout is actually the optimal solution, given that for the fixed set of given layers and the node structure, there is no other relative ordering of nodes that produces a solution with less

crossings. As explained in the methodology section, the layer for each node is fixed, which leads to layouts that contain structures like this impossible to solve for a zero crossings case, and they are therefore referred to as non-level-planar.

This highlights a major limitation of the algorithm: when a network contains non-level-planar sub-structures, like the one found in the present case, the layout will contain unavoidable crossings for a fixed set of layers. In such situations (in which the underlying graphs contain crossings), the system warns the user that the representation is invalid and can be configured to either generate or omit the diagrams.

5.2.7 Case Study 7: Large Network

The largest case study examines a 2282 node network, located in Huyton, east of Liverpool, which serves 844 customers through 5 transformers. This network contains a large number of meshed connections and, therefore, serves as the most demanding benchmark for the algorithm, allowing to test its capabilities and limitations.

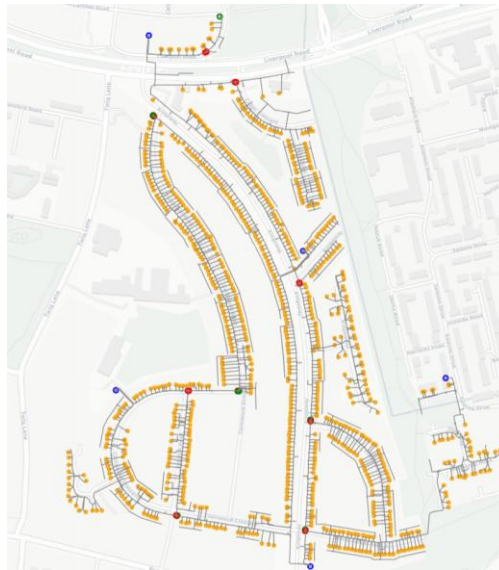


Figure 5-17: Case study 7 network GIS representation.

Output:

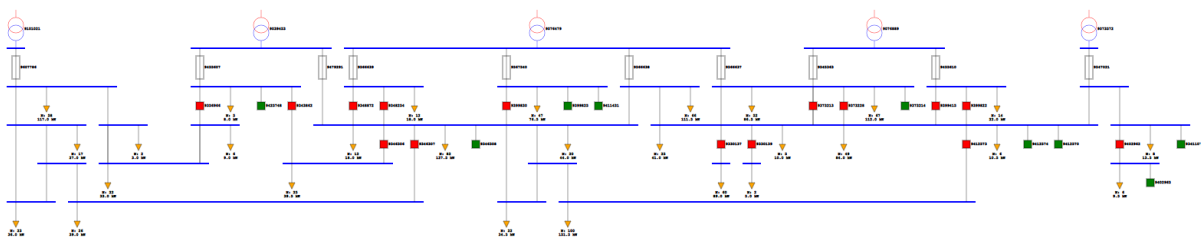


Figure 5-18: Case study 7 output diagram.

Graph laid out with 8 crossings, achieved after 77 iterations
 Number of iterations 100
 Total Node Count: 2282
 Consumer Node Count: 844
 Total Nodes Laid Out: 126
 Meshed Node Count: 11
 Runtime: 21.181025505065918 seconds

Results

The algorithm was unsuccessful in finding a crossing-free layout, with 8 crossings being the best solution it could find after 77 iterations, causing significant overlapping between different elements in the diagram.

The underlying bus layout can be observed in Figure 5.19. As in case study 6, the layout contains several non-level-planar sub-structures, so there exists no zero-crossing solution for the given set of layers. These structures therefore make the algorithm unable to produce a valid diagram.

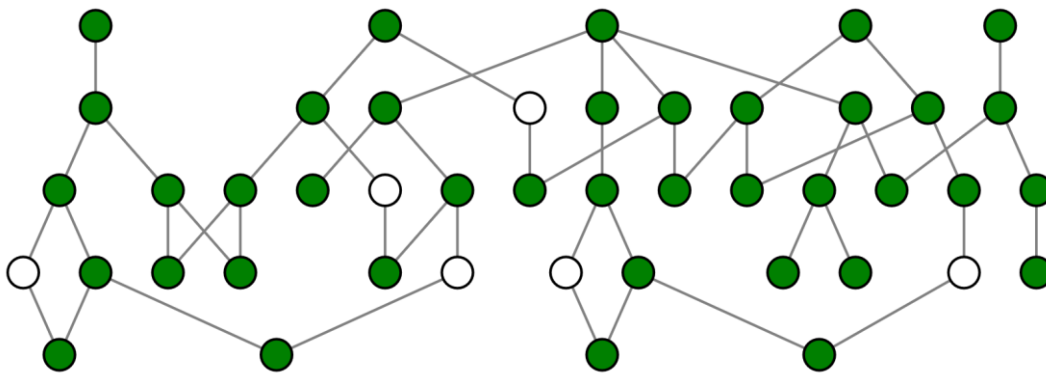


Figure 5-19: Underlying non-level-planar graph of network 7.

The 21.18 second runtime illustrates how computational cost escalates exponentially when the number of nodes and meshed connections in the network increases. This is also impacted by the algorithm being run the maximum 100 iterations, among other factors.

Overall, the results for this case study also demonstrate the practical limitations of the algorithm, which is not able to find solutions that might require dynamically adjusting the layer of each node to create a level-planar layout and subsequently find a zero-crossings solution.

5.2.8 Case Study 8: MV Network

Although the diagram representation of MV networks falls outside the formal scope of this project, a brief exploratory case study has been conducted to test the algorithm's versatility. The network being tested is a 11kV rural network located in East Lothian, Scotland,

It includes 128 MV/LV transformers, 2 HV/MV transformers, and several switches and protection devices which, for simplicity, are treated and represented as LV link nodes by the algorithm.

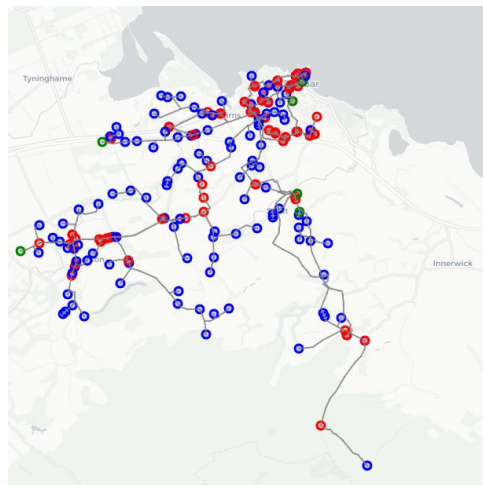


Figure 5-20: Case study 8 network GIS representation.

Results

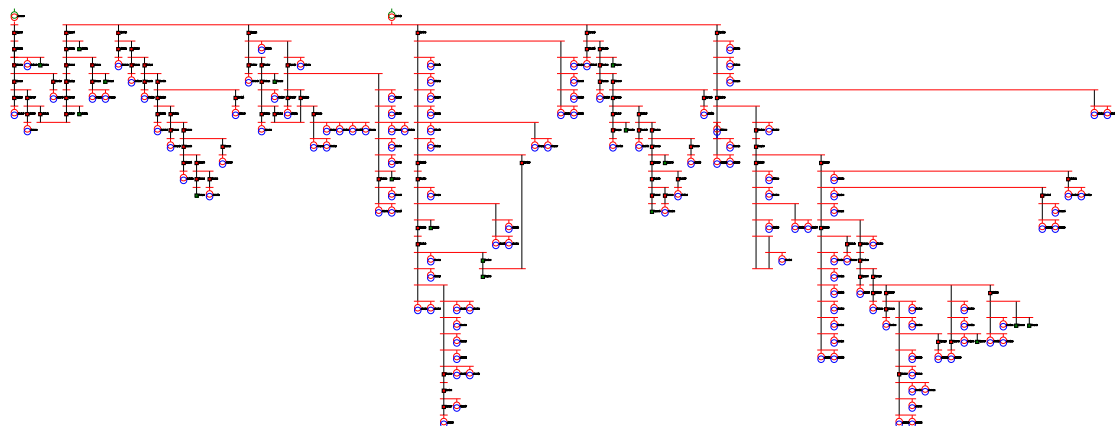


Figure 5-21: Case Study 8 output diagram.

Graph laid out with 0 crossings, achieved in 27 iterations
Number of iterations 27
Total Node Count: 1810

Consumer Node Count: 0
Total Nodes Laid Out: 643
Meshed Node Count: 7
Runtime: 17.867753505706787 seconds

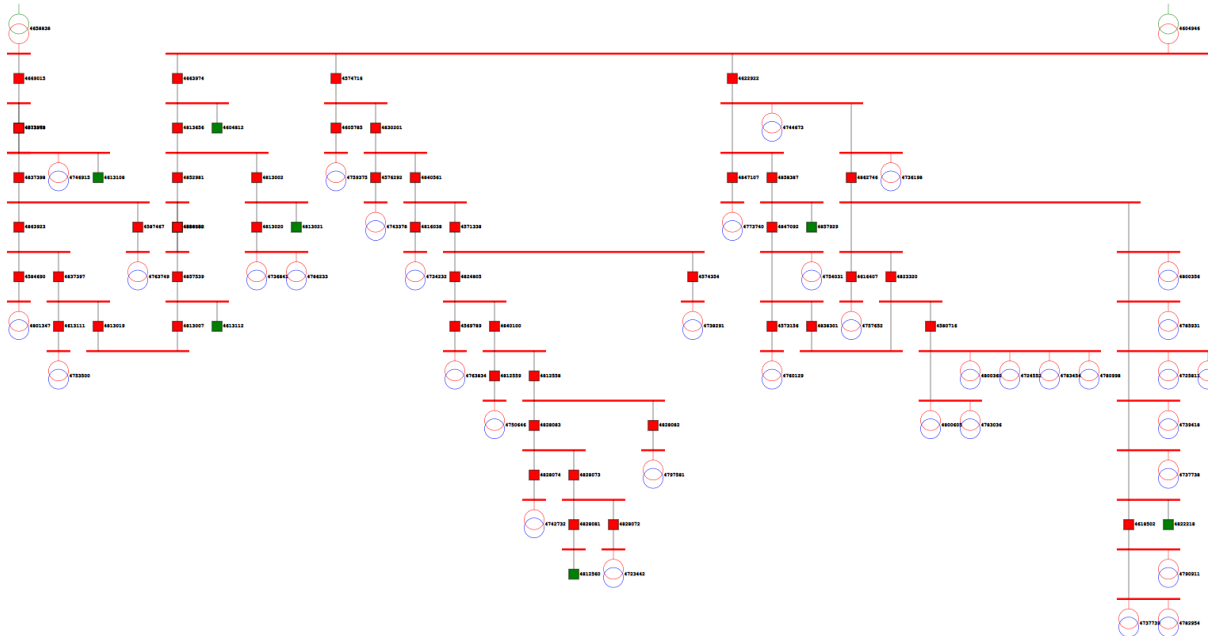


Figure 5-22: Case study 8 output diagram, zoomed-in view.

Conclusions

The algorithm was able to successfully generate the crossing-free network diagram, despite its large size and meshed topology, having found the optimal solution after 27 iterations, and a total runtime of 17.87 seconds.

Because the original network does not include any consumer nodes, and due to the large number of switches and distribution transformers it contains, the simplification and post-processing algorithms only reduced the node count from 1810 to 643 nodes, hence the large size of the resulting diagram.

This case study shows that the algorithm is versatile, and capable of producing crossing-free layouts for large and topologically complex networks even outside of its main scope around LV networks. With input parameter tuning and minor adjustments, the algorithm can be redirected to successfully generate diagrams for other sources of network data. These results also prove that as long as the graphs along with the fixed set of layers are level-planar, the algorithm is able to find solutions for complex configurations.

6 Results and Discussion

6.1 Summary of Results and Key Findings

Table 6.1 contains a summary of each case studies' results, along with the characteristics of the input networks.

Table 6-1: Summary of case studies results.

	Input		Results			
	NNODES	M	k	N_iter	N _L	Runtime (s)
Case Study 1	69	0	0	1	13	0.1
Case Study 2	145	1	0	1	15	0.13
Case Study 3	239	2	0	1	36	0.58
Case Study 4	611	2	0	1	67	1.57
Case Study 5	1126	5	0	4	64	5.07
Case Study 6	1233	4	2	20	90	6.43
Case Study 7	2282	11	8	77	127	21.18
Case Study 8	1810	7	0	27	643	17.87

The algorithm generated valid diagrams for 6 of the 8 networks studied, as it was able to find zero crossing layouts for their underlying graphs. For case studies 6 and 7 the algorithm was not able to find a zero crossings solution for their underlying graphs and therefore generated invalid diagrams. Runtime increased significantly for larger and more meshed graphs.

The obtained results validate the following key findings:

Valid diagrams for most cases, failures align with expectations. The first 4 case studies produced a successful layout with zero crossings on the first iteration, reflecting the algorithm's effectiveness in small-sized, low-meshed networks. For case study 5, a significantly more meshed network, the algorithm also produced a valid diagram, although it required several layout iterations. The algorithm was unable to generate a zero-crossings solution for case studies 6 and 7, and therefore, a valid final diagram. However, this aligns with the expected result, as both networks have an underlying non-level planar graph which is generated by the fixed layer assignment process.

Runtime scales with size and meshedness. Runtime increased significantly in relation to the number of nodes in the graph, remaining almost linear for less complex networks (only ~0.1

seconds for the simplest cases), but becoming exponential as the network complexity, both in size and meshedness, increased, up to 21.18 s for the most complex network. This is caused by the longer graph traversals required for several algorithms, as well as the larger number of iterations of the layout algorithms. A smaller number of traversals (by for example integrating several algorithms together) could potentially lower the runtime for larger networks.

Input parameter effects are predictable and controllable. Consumer grouping parameters tune horizontal and vertical detail as intended in the final diagram. Decreasing *max_consumers_per_bus* increases vertical resolution, showing more detail into the network's topology by displaying more buses, while decreasing *max_consumers_per_group* allows to show more horizontal grouping granularity, showing higher numbers of consumer groups connected at the same network level.

MV network case study validates generality. The algorithm was also successful in producing a valid diagram for a MV network, out of the formal project's scope, demonstrating the proposed pipeline is generalizable to successfully generate valid diagrams for networks with significantly different topological characteristics than the LV networks on which it is mainly implemented.

Randomized naïve iterations of the layout algorithm improve results. Re-running the crossing minimization section of the layout algorithm with randomized inputs allowed the algorithm to escape local minima and find optimal solutions for case studies 5 and 8, whose solution was not found on the first algorithm iteration. For case studies 6 and 7, even though it could not find a solution with no crossings due to the non-level-planarity of the graphs, later iterations also successfully reduced the number of crossings. This performance is further analyzed in the following section.

6.2 Layout Algorithm Performance Analysis

To visualize and benchmark the performance of the layout algorithm, a simulation for each of the three most demanding case studies performed has been conducted, in which the layout algorithm exclusively was run 30 times with different values for the number of random naïve iterations, to test how increasing the maximum number of iterations can lead the algorithm to find more optimal solutions, and how the execution time increases in relation to the number of iterations.

Each test was conducted using one of the following values of maximum iterations, for the networks in case studies 6, 7 and 8:

$N_max_iterations = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 150, 200, 250, 300]$

The results can be observed in Figure 6.1:

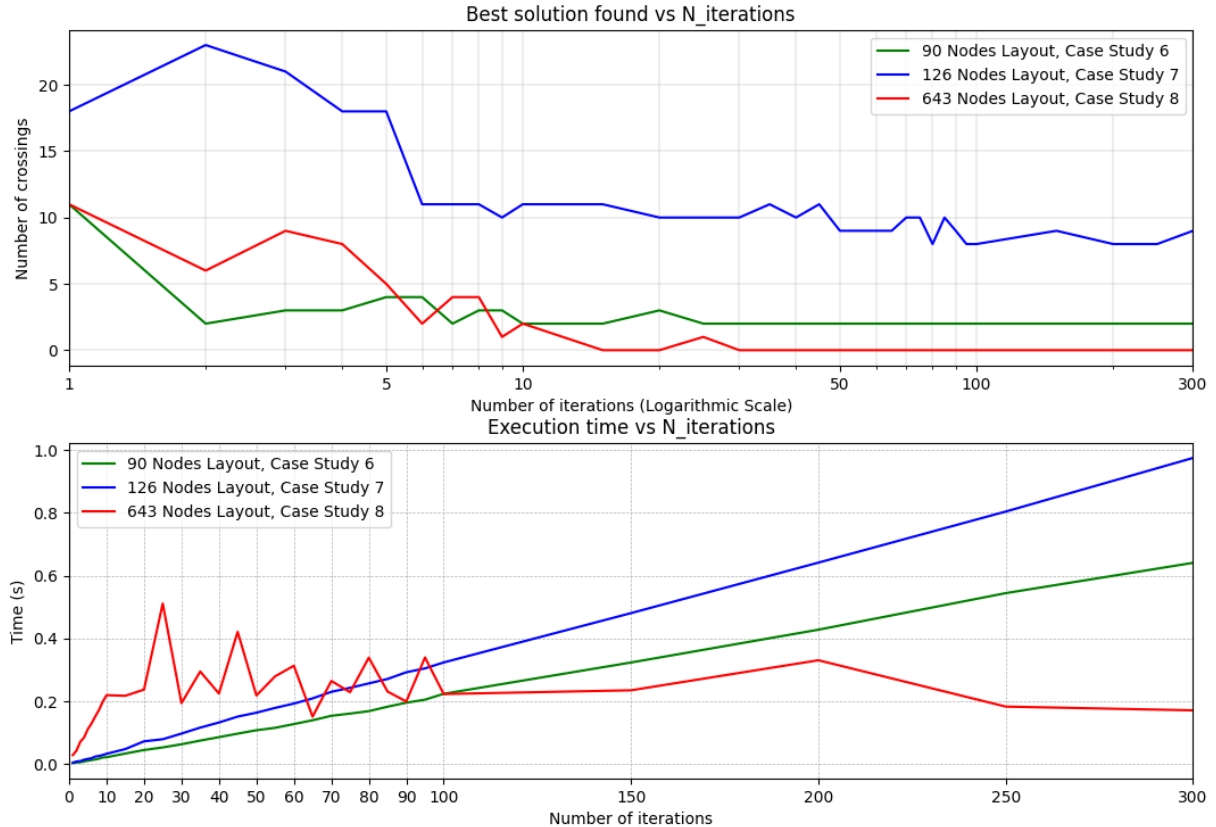


Figure 6-1: Layout algorithm performance, case studies 6, 7 and 8.

As the figures illustrate, the tests for each network produced significantly different results. Starting with the network used in case study 6, the smallest out of the ones tested, the algorithm found the best solution (2 crossings) in only two iterations. However, due to the random nature of the algorithm, in the following tests it did not converge to that optimal solution until the 7th test, when 7 max iterations were used. As expected, for higher iteration tests, the algorithm always managed to find the best solution, which is the corresponding to two crossings, as explained in the case study. Execution time increased linearly, with the number of iterations, as expected, but was always shorter than 1 second, indicating that the layout optimization phase represents only a small fraction of total time to run the algorithm, which in case of case study 6 was 6.42 seconds for 100 max iterations.

In contrast, the results for the tests done on the larger and more complex case study 7 network, showed that the algorithm required a significantly larger number of max iterations to find the best solution it found among all tests (8 crossings), around 80. It also shows a greater variance in the results for higher max iteration values, which highlights that the randomness of the algorithm is correlated with the complexity of the network, suggesting that a higher number of maximum iterations should be used to maximize the odds of obtaining the optimal solution. Runtime also scaled linearly, with similar values to the case study 6 network, but a higher slope which is likely caused by the increased complexity and size of the network.

Finally, the results for the HV network used in case study 8 show significant differences, as in this case the algorithm did find a zero crossings solution. This crossing-free solution was first found in the 15 max iterations test, and the algorithm managed to consistently find it when the number of max iterations was higher than 30. This proves that when the underlying network contains no non-level-planar substructures, the algorithm can effectively find the optimal solution within a small number of iterations. The execution time figure shows a significant variability after 10 max iterations because the algorithm stops after finding the 0 crossings solution, leading to random total execution times for runs that are able to find that solution, and therefore uncorrelated with the number of max iterations. For tests that did not converge to the optimal solution (max iterations ≤ 10 and $= 25$), the algorithm did have a runtime proportional to max iterations, and its slope can be observed to be significantly higher than in the two previous simulations, further proving that the slope is correlated with the size of the network laid out.

6.3 Method Limitations and Design Trade-offs

The results of the case studies also highlight several limitations and trade-offs

Fixed layer assignment vs level planarity. The algorithm assigns node layers based solely on distance from the roots, before crossing minimization. This fixed layer assignment provides simplicity and creates intuitive layers, that allow to easily represent topological distance from the transformer. This however causes a significant limitation, that the creation of non-level-planar structures is not avoided. An algorithm that dynamically checks level planarity and adjusts the layers to ensure it would be necessary to solve this, but at the cost of increased complexity and without certain success, as it is another NP-hard problem.

Naïve heuristic approach vs deterministic solution. The success of the crossing minimization algorithm relies on the naïve heuristic algorithm being able to escape local optima to find the optimal solution, which is increasingly unlikely as network size increases and requires more iterations. The approach is effective for simple networks, but more complex cases might benefit from a more deterministic approach, through MILP, for example, although this would likely increase runtime significantly.

Limited parametric tuning. The input parameters allow to adjust the maximum number of consumers in each bus and group, up to a certain degree. The number of consumers on each bus in some cases can be larger than the specified threshold due to several consumers being connected at the same topological point, or due to the graph simplification and normalization process, limiting the capability of tuning the final diagram through changing the input parameters. Additionally, when several groups of consumers are created in the same bus node, the groups generally do not reflect clusters consumers close to each other, which could be implemented through a more complex algorithm that takes this into consideration when creating clusters.

Exponential time complexity for complex networks. Although for the majority of cases studied, the algorithm was able to generate the single line diagrams in reasonably short times (less than 5 seconds), it has been shown that for more complex networks runtime increases exponentially. This might make the algorithm unsuitable for increasingly meshed and complex graphs.

7 Conclusions and Future Work

This thesis demonstrates and delivers a reproducible pipeline that can be used to translate LV network GIS data into standard, readable, single-line diagrams that faithfully represent the underlying topology of the network. The separation of graph simplification, layout and symbol plotting enables a modular approach that can be tuned via several inputs to fit the data characteristics and to adjust characteristics of the final diagram, which contains a simplified view of the underlying network topology and key characteristics. The method produced six valid diagrams for eight diverse real networks studied, with the two unsuccessful cases explained by level-planarity violations caused by the fixed layer assignment algorithm. The runtime for these cases increases linearly with the number of nodes for the least complex networks but rises exponentially for larger and more meshed networks, although remaining on a timescale suitable for interactive or near interactive use on personal computing hardware.

Some suggested areas for further development and potential result improvement include:

- Substitute the fixed layer assignment and heuristic crossing minimization for a **MILP approach** that aims to solve a two-objective problem, jointly optimizing the layer assignment and relative position of nodes inside each layer to reliably produce valid zero-crossings layouts, not constrained by fixed layer assignments which might be non-level-planar. However, this approach, if feasible, might have an excessive time complexity that could make it impractical.
- **Heuristic approaches to adjust the layer assignment.** Detection of level planarity is complex, however, iteratively detecting smaller non-level planar structures inside a larger graph might be feasible, and changing the layer of some nodes in the structure while checking level planarity iteratively might be a valid heuristic approach for creating level planar graphs that can be laid out using the current crossing minimization and layout algorithms.
- **Pre-computing layouts for different values of *max_consumers_per_bus*.** As changing the value of max consumers can change the resolution into network details, effectively changing the number of bus nodes in some network sections, this can potentially cause non-level planar structures to disappear as the layers change. Although this is not an optimal solution as it removes the capability of tuning the input parameters,

it might allow to generate single line diagrams of some networks that could not be generated using a different *max_consumers_per_bus* parameter. Pre-computing the layouts with several parameter values could allow storing what values produce suitable diagrams.

- **Interactive final diagram.** From the final layout, an interactive implementation of the diagram could be created, through a web-based application for example, that allows features like selecting elements to view more detailed information, selecting two points to create traces between elements in the network, or visualizing with more detail certain consumer clusters that the operator selects, for example.
- **Partitioning the layout.** For larger networks for which it might not be feasible to generate a single line diagram without components overlapping (due to complexity and non-level-planarity, a possible solution could be to compute several smaller layouts independently and later generating the complete layout by unifying the smaller sections. This would however require considering several constraints, like keeping boundary connection sections on the edges of the smaller sub-layouts, so that when creating the unified layout later, no connections overlap with other network elements.
- **Geographical element anchoring.** The pipeline could be re-focused to generate a hybrid representation of the network that retains relative geographical positions for some elements like transformers, while implementing diagram representations of other sections like clusters of consumers. This would require a significantly modified layout and plotting algorithm that also takes as an input the geographical positions of different elements, and whether it is desired to keep the corresponding element anchored to each position.
- **Generalized API.** Provide a more generalized API to tune model parameters and labels, improving compatibility and easing adaptation to additional data sources.

8 References

- [1] S. Gordon, C. McGarry, and K. Bell, “The growth of distributed generation and associated challenges: A Great Britain case study,” *IET Renewable Power Generation*, vol. 16, no. 9, pp. 1827–1840, Jul. 2022, doi: 10.1049/rpg2.12416.
- [2] KOZO SUGIYAMA, SHOJIRO TAGAWA, and MITSUHIKO TODA, “Methods for Visual Understanding of Hierarchical System Structures,” 1981.
- [3] Michael Benson, Iona Stewart, Georgina Hutton, and Dr Roger Tyers, “Electric Vehicles and Infrastructure (CBP-7480),” 2025.
- [4] I. Diahovchenko, G. Morva, A. Chuprun, and A. Keane, “Comparison of voltage rise mitigation strategies for distribution networks with high photovoltaic penetration,” *Renewable and Sustainable Energy Reviews*, vol. 212, p. 115399, Apr. 2025, doi: 10.1016/J.RSER.2025.115399.
- [5] Z. Cheng, E. Udren, J. Holbach, M. J. Reno, and M. E. Ropp, “Protection and Control Challenges of Low-Voltage Networks with High Distributed Energy Resources Penetration - Part 1: Utility Workshop and Low-Voltage Network Modeling,” in *2023 76th Annual Conference for Protective Relay Engineers, CFPR 2023*, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/CFPR57837.2023.10126827.
- [6] IEEE Standards Association, *IEEE Std 1547-2018 (Revision of IEEE Std 1547-2003) : IEEE Standard for Interconnection and Interoperability of Distributed Energy Resources with Associated Electric Power Systems Interfaces*. IEEE, 2018.
- [7] H. Lee Smith, “A Brief History of Electric Utility Automation Systems,” *EE Online*.
- [8] M. T. Fischer and D. A. Keim, “Towards a Survey of Visualization Methods for Power Grids,” Apr. 2022, [Online]. Available: <http://arxiv.org/abs/2106.04661>
- [9] T. J. Overbye and J. D. Weber, “Visualization of Power System Data,” 2000.

- [10] T. J. Overbye, E. M. Rantanen, and S. Judd, "Electric Power Control Center Visualization using Geographic Data Views," 2007.
- [11] P. C. Wong *et al.*, "A novel visualization technique for electric power grid analytics," *IEEE Trans Vis Comput Graph*, vol. 15, no. 3, pp. 410–423, May 2009, doi: 10.1109/TVCG.2008.197.
- [12] P. C. Wong, H. Foote, P. Mackey, G. Chin, Z. Huang, and J. J. Thomas, "A space-filling visualization technique for multivariate small-world graphs," *IEEE Trans Vis Comput Graph*, vol. 18, no. 5, pp. 797–809, 2012, doi: 10.1109/TVCG.2011.99.
- [13] D. Owerko, F. Gama, and A. Ribeiro, "Unsupervised Optimal Power Flow Using Graph Neural Networks," Oct. 2022, [Online]. Available: <http://arxiv.org/abs/2210.09277>
- [14] A. R. Singh, R. S. Kumar, M. Bajaj, C. B. Khadse, and I. Zaitsev, "Machine learning-based energy management and power forecasting in grid-connected microgrids with multiple distributed energy sources," *Sci Rep*, vol. 14, no. 1, Dec. 2024, doi: 10.1038/s41598-024-70336-3.
- [15] PNNL, "ChatGrid™: A New Generative AI Tool for Power Grid Visualization," <https://www.pnnl.gov/news-media/chatgridtm-new-generative-ai-tool-power-grid-visualization>.
- [16] M. Zhou, J. Yan, and Q. Wu, "Graph Computing and Its Application in Power Grid Analysis," *CSEE Journal of Power and Energy Systems*, vol. 8, no. 6, pp. 1550–1557, Nov. 2022, doi: 10.17775/CSEEJPES.2021.00430.
- [17] A. Monticelli, *State Estimation in Electric Power Systems*. 1999.
- [18] E. Baran and F. F. Wu, "NETWORK RECONFIGURATION IN DISTRIBUTION SYSTEMS FOR LOSS REDUCTION AND LOAD BALANCING," 1989.
- [19] 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. 7269–7276, Nov. 2018, doi: 10.1109/TPWRS.2018.2854172.

- [20] N. Rezaei, A. Roosta, N. Rezaee, M. Nayeripour, A. Roosta, and T. Niknam, "Role of GIS in Distribution Power Systems," 2009. [Online]. Available: <https://www.researchgate.net/publication/265041877>
- [21] Vahraz Zamani and Terry Nielsen, "How to Create an Accurate Network Model and Dynamic State Data for an Advanced Distribution Management System (ADMS)," *IEEE Smart Grid*, 2020.
- [22] Internet Engineering Task Force (IETF), "RFC 7946 - The GeoJSON Format," 2016.
- [23] ArcGIS, "GeoJSON," 2025.
- [24] A. Bosisio, A. Berizzi, M. Merlo, A. Morotti, and G. Iannarelli, "A GIS-Based Approach for Primary Substations Siting and Timing Based on Voronoi Diagram and Particle Swarm Optimization Method," *Applied Sciences (Switzerland)*, vol. 12, no. 12, Jun. 2022, doi: 10.3390/app12126008.
- [25] H.-Y. Wu, B. Niedermann, S. Takahashi, and M. Nöllenburg, "A Survey on Computing Schematic Network Maps: The Challenge to Interactivity," Aug. 2022, [Online]. Available: <http://arxiv.org/abs/2208.07301>
- [26] S. Anand, S. Avelar, J. M. Ware, and M. Jackson, "Automated schematic map production using simulated annealing and gradient descent approaches," 2007.
- [27] D. Chivers and P. Rodgers, "Exploring Local Optima in Schematic Layout," 2013.
- [28] Y.-S. Jelmg, L.-G. Chen, and T.-M. Parng, "ASG: Automatic schematic generator," 1991.
- [29] Z. Dong, W. Cao, M. Zhang, D. Tao, Y. Chen, and X. Zhang, "CktGNN: Circuit Graph Neural Network for Electronic Design Automation," Feb. 2024, [Online]. Available: <http://arxiv.org/abs/2308.16406>
- [30] D. Toral and G. A. Alonso-Concheiro, "OPTIMAL AUTOMATIC DRAWING OF ONE-LINE DIAGRAMS Canales-Ruiz, fMrber ITEE."
- [31] 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, doi: 10.1016/j.epsr.2003.12.005.

- [32] P. Healy and N. S. Nikolov, "Hierarchical Drawing Algorithms," 2013.
- [33] J. X. Ruixing Lin and Honggeng Yang, *Intelligent Automatic Layout of One-line Diagrams for District Electrical Distribution Network*. IEEE, 2010.
- [34] Peng Sen, Liu Shi-jin, You Feng, Cheng Wei, Cheng-long, and Zheng Hao-quan, "Automatic Generation and Incremental Update Method of Single Line Diagram of Distribution Network," *Proceedings of the 7th PURPLE MOUNTAIN FORUM on Smart Grid Protection and Control*, 2023.
- [35] Boxi Zhou, Lianxi Sun, Hongwei Zhang, Yongfei Yin, Wei Ding, and Wenzhen Huang, "Automatic Single-line Diagram Generation of Distribution Network with Rings Based on GA," 2016.
- [36] C. Yang, S. Wu, T. Liu, Y. He, J. Wang, and D. Shi, "Efficient generation of power system topology diagrams based on Graph Neural Network," *Eng Appl Artif Intell*, vol. 149, p. 110462, Jun. 2025, doi: 10.1016/J.ENGAPPAI.2025.110462.
- [37] DIgSILENT, "PowerFactory: Network Diagrams and Graphic Features," <https://www.digsilent.de/en/network-diagrams-and-graphic-features.html>.
- [38] ETAP, "Intelligent Single-Line Diagram," <https://etap.com/product/intelligent-electrical-one-line-diagram>.
- [39] Siemens, "OpenDSS," <https://www.siemens.com/global/en/products/energy/grid-software/planning/pss-software/pss-e.html>.
- [40] Robin J. Wilson, *Introduction to Graph Theory*, Fourth edition. Longman Group Ltd, 1972.
- [41] Paul E. Black, "Dictionary of Algorithms and Data Structures," 2022, NIST.
- [42] E. M. Reingold and J. S. Tilford, "Tidier Drawings of Trees," 1981.
- [43] Kay Jan Wong, "Reingold Tilford Algorithm Explained With Walkthrough," <https://towardsdatascience.com/reingold-tilford-algorithm-explained-with-walkthrough-be5810e8ed93/>.

- [44] G. Brückner and I. Rutter, “Partial and constrained level planarity,” *Theor Comput Sci*, vol. 1045, p. 115291, Aug. 2025, doi: 10.1016/J.TCS.2025.115291.
- [45] Python Software Foundation, “Python Programming Language, Version 3.13.1,” Dec. 2024, *Python Software Foundation, Wilmington, DE*. [Online]. Available: <https://www.python.org/downloads/release/python-3131/>
- [46] A. A. Hagberg hagberg, lanlgov -Los, D. A. Schult, and P. J. Swart swart, “Exploring Network Structure, Dynamics, and Function using NetworkX,” 2008. [Online]. Available: http://conference.scipy.org/proceedings/SciPy2008/paper_2
- [47] T. pandas development team, “pandas-dev/pandas: Pandas,” Jul. 2025, *Zenodo*. doi: 10.5281/zenodo.15831829.
- [48] Inc. Graphistry, “Graphistry Sugiyama Layout,” <https://pygraphistry.readthedocs.io/en/latest/api/layout/sugiyama.html>.
- [49] C. R. Harris *et al.*, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020, doi: 10.1038/s41586-020-2649-2.
- [50] K. Jordahl *et al.*, “geopandas/geopandas: v0.8.1,” Jun. 2025, *Zenodo*. doi: 10.5281/zenodo.15750510.
- [51] python-visualization, “Folium.” [Online]. Available: <https://python-visualization.github.io/folium/>
- [52] P. T. Inc., “Collaborative data science,” 2015, *Plotly Technologies Inc., Montreal, QC*. [Online]. Available: <https://plot.ly>
- [53] A. Research, “Graphviz - Graph Visualization Software,” 2008. [Online]. Available: <http://www.graphviz.org/>
- [54] Manfred Moitzi, “Svgwrite Documentation,” <https://svgwrite.readthedocs.io/en/latest/index.html>.

9 Appendix I: Alignment with the UN Sustainable Development Goals

The project contributes to sustainable development by helping utilities and system operators better understand their electric distribution networks. Automatically generating single-line diagrams from raw GIS data, enables, more efficient grid decisions, and enables better observability at the distribution level. The result is a practical tool that system operators can adopt within existing tools to improve reliability, integrate clean generation, and plan upgrades with fewer resources. The project aligns more specifically with the following SDGs:

SDG 7: Affordable and Clean Energy. The method supports wider access to reliable and modern energy services by giving planners and operators an immediate picture of the topological characteristics of distribution networks. These diagrams speed up studies for connecting rooftop solar, community batteries, electric vehicle chargers and heat pumps, located at the distribution level. Additionally, the greater observability provided by schematics is essential for the operation of active distribution networks due to the instability caused by the growing integration of DERs. In short, better visibility enables more affordable operations and smoother integration of clean energy.

SDG 9: Industry, Innovation and Infrastructure. The effective development of modern infrastructure depends on access to high-quality information. Automating the production of single-line diagrams replaces a slow, manual process with a repeatable and scalable one. The pipeline can be implemented to guide the decisions of planning engineers and improve the efficiency and resilience of distribution networks. Improved grid infrastructure is essential for all countries, but especially for developing ones, which can obtain improved returns through more efficient investments.

SDG 13: Climate Action. Climate goals require faster electrification and smarter grids. By minimizing the effort to map and understand distribution networks, the project allows improvements to DER management and network planning, through greater observability, which are both key in the transition to 100% clean energy power systems.