FORWARD: A Feasible Radial Reconfiguration Algorithm for Multi-Source Distribution Networks ⋆

Joan Vendrell Gallart [a], Russell Bent [b], Solmaz Kia [a]

[a] *University of California Irvine, Irvine, California, USA*

[b] *Los Alamos National Laboratory, Los Alamos, New Mexico, USA*

Abstract

This paper considers an optimal radial reconfiguration problem in multi-source distribution networks, where the goal is to find a radial configuration that minimizes quadratic distribution costs while ensuring all sink demands are met. This problem arises in critical infrastructure systems such as power distribution, water networks, and gas distribution, where radial configurations are essential for operational safety and efficiency. Optimal solution for this problem is known to be NP-hard. In this paper, we prove further that constructing a feasible radial distribution configuration is weakly NP-complete, making exact solution methods computationally intractable for large-scale networks. We propose `FORWARD` (Feasibility Oriented Random-Walk Inspired Algorithm for Radial Reconfiguration in Distribution Networks), a polynomial-time algorithm that leverages graph-theoretic decomposition and random walk principles to construct feasible radial configurations. Our approach introduces novel techniques including strategic graph partitioning at articulation points, dual graph condensation to address greedy shortsightedness, and capacity-aware edge swapping for infeasibility resolution. We provide rigorous theoretical analysis proving feasibility guarantees and establish a compositional framework enabling parallel processing while preserving optimality properties. Comprehensive numerical evaluation on networks ranging from IEEE standard test systems to 400-node small-world networks demonstrates that `FORWARD` consistently outperforms commercial MINLP solvers, achieving optimal or near-optimal solutions in seconds where traditional methods require hours or fail entirely. The algorithm's polynomial-time complexity and scalability make it particularly suitable for real-time distribution network management and as an effective initialization strategy for iterative optimization solvers.

Key words: Radial Reconfiguration; Network flow problem; Graphs; Power Systems.

1 Introduction

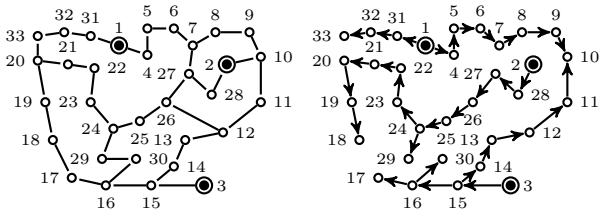
In today's rapidly evolving technological and infrastructural landscape, distribution networks, encompassing electricity, gas, and water irrigation systems, are fundamental to resource delivery across diverse geographical areas. In these distribution networks, there is often a need for radial reconfiguration (see Fig. 1), driven by the complex physical properties and behaviors inherent in resource flow systems, as well as the necessity to adhere to engineering and safety standards. Radial configurations help minimize losses and ensure system stability, which are essential for both economic viability and environmental sustainability [21,1,16]. With the increasing complexity of modern distribution systems, there is a need for computationally fast solutions to obtain reliable resource distribution.

Optimizing distribution over all the possible radial reconfiguration often is cast as Mixed-Integer Non-Linear Programming (MINLP) problems [16,32,31]. The complexity of solving these optimization problems arises from the exponential number of possible configurations, particularly as the network scales, making it an NP-hard problem [24]. Traditional MINLP methods often fall short of delivering an optimal solution due to their computational demands, especially in large-scale networks. Recent advancements have shifted towards suboptimal algorithms that can efficiently handle this complexity. For instance, leveraging graph theory and random walk processes has shown promise in simplifying the search space and reducing computational effort [23].

This paper constructs a polynomial-time feasible solution for a class of optimal radial reconfiguration problem in a distribution network with multiple sources

⋆ A preliminary version of this paper was presented in 2025 American Control Conference [28].

Email addresses: `jvendrel@uci.edu` (Joan Vendrell Gallart), `rbent@lanl.gov` (Russell Bent), `solmaz@uci.edu` (Solmaz Kia).



(a) Original network. (b) Radial configuration.

Fig. 1. In the optimal reconfiguration problem, the highlighted nodes in dark are the sources and the remaining nodes are the sinks. In radial configuration, some nodes, e.g., sink node 10 may receive receive input from two different edge; despite that there is no cycle in the graph. The network used here is the IEEE 33 network [9].

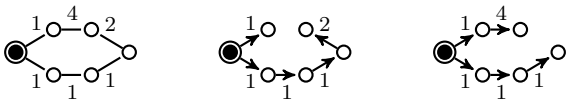
and sinks. Our method, termed as FORWARD: **F**easibility **O**riented **R**andom-**Wa**lk Inspired Algorithm for **R**adial Reconfiguration in **D**istribution Networks, leverages the graph-like characteristics of distribution systems and takes into account how potential flows naturally navigate through a network. Our algorithm uses a greedy radial construction process starting from source nodes, incrementally adding edges while considering the flow across constructed components and the demanded output of the remaining nodes to reach. We draw inspiration from the similarities between electric flow in power networks and random walks [10], developing a novel 'sampling' method [1] for constructing radial configurations. In the random walk approach to describe electricity distribution in a given network, a weight proportional to the inverse of the resistance along the corresponding link in the power network is assigned to each link. This creates a notion of the edge weights being the conductance of the edge. Just as electricity flows through paths of least resistance, a random walk probabilistically selects paths based on transition probabilities influenced by edge weights. FORWARD respects capacity constraints and ensures the network remains feasible and efficient by reducing the sample space to feasible edges and prioritizing critical edges.

We introduce an innovative mechanism (Net-Concad function explained in Section 6.1.3) in our incremental 'sampling' process that addresses the shortsightedness of greedy selections by informing the process of the comprehensive demand of the remaining nodes. We carry out a rigorous evaluation of the algorithm, including its formal feasibility guarantees. Extensive numerical experiments demonstrate the algorithm's efficacy across various distribution networks, highlighting its potential for real-world applications [12].

In summary, the main contributions of this work:

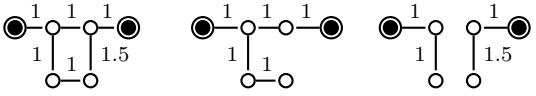
- We prove that the problem of constructing a feasible radial configuration for multi-source distribution networks is weakly NP-complete, a new theoretical result that highlights the computational intractability of the problem and motivates the need for efficient algorithms.
- We propose a novel polynomial-time algorithm, named FORWARD, that leverages a compositional framework to guarantee feasible solutions and rapidly find high-quality solutions with a time complexity of $\mathcal{O}(n^2 \log n)$ on sparse networks.
- We introduce several key algorithmic innovations, including a dual graph condensation technique to overcome greedy limitations and a capacity-aware edge swapping method to resolve infeasibilities.
- We demonstrate that FORWARD consistently outperforms commercial MINLP solvers on large-scale networks (up to 400 nodes), achieving optimal or near-optimal solutions in seconds where other methods fail or take hours, making it highly suitable for real-time network management.

[1] Although our radial configuration construction is deterministic, we use 'sampling' as a conceptual analogy.



(a) $\mathcal{G}_D$ (b) MST (c) $\mathcal{G}(\mathcal{V}_D, \mathcal{S}^\star)$

Fig. 2. Example where radial distribution constructed from MST (plot (b)) is not the minimum radial configuration, $\mathcal{G}(\mathcal{V}_D, \mathcal{S}^\star)$, (plot(c)). This phenomenon is due to the quadratic nature of the cost; let the demand at each sink be d. The source node, highlighted in bold, can supply input $5d$. The edge weights are shown on the edges. In the feasible radial distribution network (b) the cost is $1 \cdot (d)^2 + 1 \cdot (4d)^2 + 1 \cdot (3d)^2 + 1 \cdot (2d)^2 + 2 \cdot (d)^2 = 32d^2$, meanwhile in the feasible radial distribution network (c) the cost is $1 \cdot (2d)^2 + 4 \cdot (1d)^2 + 1 \cdot (3d)^2 + 1 \cdot (2d)^2 + 1 \cdot (1d)^2 = 22d^2$.



(a) $\mathcal{G}$ (b) MST (c) MSF

Fig. 3. An example where MSF results is a better outcome than MST: sources highlighted provide an input of $2d$ each and the sink nodes demand each d.

Related work: Several methods in the literature, especially in power networks applications, leverage topological properties to address the reconfiguration problem [7]. For example, [19] [29] use spectral clustering followed by local greedy search to identify radial configurations. In [17], the reconfiguration problem is linked to the maximum flow problem, based on Ford-Fulkerson's work [8], which has been extensively studied in both single- and multi-source contexts [6], [18], [4]. These approaches often include a repair procedure for unfeasible solutions, which eventually results inefficient in practice. While some methods address minimum-cost distri-

bution, radiality remains a critical constraint [13], and cycle-breaking methods used to tackle it offer limited guarantees for large graphs [5]. Other approaches, such as those based on the minimum spanning tree (MST) problem, face challenges due to the constraints of the reconfiguration problem, making the MST approach NP-hard in this context (see Fig. 2). Moreover, although algorithms like Kruskal's or Prim's [26] can find MSTs in polynomial time, they lose their optimality when multiple sources are involved, note that the minimum spanning forest (MSF) is not necessarily a subset of the MST (see Fig. 3).

2 Problem Setting

We consider a network-flow distribution problem over a bidirectional distribution network $\mathcal{G}_D = \mathcal{G}(\mathcal{V}_D, \mathcal{E}_D)$ with $|\mathcal{V}_D| = N$ nodes, $|\mathcal{E}_D| = m$ edges, a set of n_g source nodes $\mathcal{V}_g \subset \mathcal{V}_D$, and $n_c = N - n_g$ sink nodes $\mathcal{V}_c = \mathcal{V}_D \backslash \mathcal{V}_g$, each with specified input and output. We let $\mathbf{d} \in \mathbb{R}^N_{\geq 0}$ be the output vector whose non-zero elements correspond to sink nodes and $\mathbf{g} \in \mathbb{R}^N_{\geq 0}$ be the input vector whose non-zero elements belong to source nodes; i.e., $g_i \in \mathbb{R}_{>0}$ (resp. $d_j \in \mathbb{R}_{>0}$) for $i \in \mathcal{V}_g$ (resp. $j \in \mathcal{V}_c$) and $g_i = 0$ (resp. $d_j = 0$) otherwise. The assumption is that inputs match outputs, i.e., $\sum_{i \in \mathcal{V}_g} g_i = \sum_{i \in \mathcal{V}_c} d_i$.

The goal is to find an (oriented) *radial configuration* that delivers the input flow from the source nodes to the sink nodes with minimal overall cost to meet the output demands. An example valid radial configuration is shown in Fig. 1, which shows a constructed radial configuration (plot (b)) from a given distribution network (plot (a)). This cost results from the 'resistance' or 'toll' along the edges, characterized as a quadratic function of the flow across them.

Definition 1 ***(Set of radial configurations)*** *A radial configuration is a polyforest*[2] *that includes all the nodes $\mathcal{V}_D$, has roots at $\mathcal{V}_g$ and the undirected version of its edges are subset of $\mathcal{E}_D$. We denote the set of these polyforest digraphs by $\mathcal{F}(\mathcal{G}_D, \mathcal{V}_g)$. For brevity, when clear from context, we will use only $\mathcal{F}$.* □

The problem of interest is formalized as

$$\min \sum\nolimits_{(i,j)\in\mathcal{S}} C_{i,j} \cdot x_{i,j}^2, \quad \text{subject to} \tag{1a}$$
$$\mathcal{G}(\mathcal{V}_D, \mathcal{S}) \in \mathcal{F}, \tag{1b}$$
$$0 \leq \mathbf{x}(\mathcal{S}) \leq \bar{\mathbf{x}}(\mathcal{S}), \tag{1c}$$
$$A(\mathcal{S})\,\mathbf{x}(\mathcal{S}) = \mathbf{g} - \mathbf{d}, \tag{1d}$$

[2] A polyforest (or directed forest or oriented forest) is a directed acyclic graph whose underlying undirected graph is a forest. A forest is a type of graph that contains no loops. Consequently, forests consist solely of trees that might be disconnected, leading to the term 'forest' being used [20].

where $x_{i,j} \in \mathbb{R}_{>0}$ is the flow across the link (i, j), $A(\mathcal{S})$ is the incidence matrix of the radial configuration $\mathcal{S}$ (a decision variable of the optimization problem), $C_{i,j}$ is the coefficient of the cost of edge $(i, j) \in \mathcal{S}$. The constraint (1b) confines the radial configuration to $\mathcal{F}$, (1c) enforce capacity across each edge in $\mathcal{S}$, and (1d) enforces the flow conservation (Kirchhoff's law) at the nodes. We consider the following assumption across this paper.

Assumption 1 *(Feasibility and feasible radial distribution configuration): The feasible solution set of the optimization problem* (1) *is non-empty. We refer to any feasible solution as a* feasible radial distribution configuration. □

This assumption means that at least there is one feasible radial distribution configuration, for which the inputs can meet the specified output, despite the capacity bounds.

Problem (1) is highly relevant to various potential flow applications, including natural gas, water, and electricity distribution networks. For instance, in power systems, the efficient and reliable distribution of energy has become increasingly critical with the growing integration of renewable energy and distributed generation sources. Modern power distribution networks, comprising multiple distributed generators, must operate in a radial configuration to adhere to engineering and safety standards. Minimizing energy loss within these networks is vital for both economic viability and environmental sustainability. Consequently, radial configurations cannot be chosen arbitrarily; they must be designed to ensure feasibility and to optimize energy loss (cost of operation).

3 Notations

In this section we introduce our essential notation and review the definition of some graph theoretic and algorithmic concepts and tools we use in the paper:

- $\mathcal{S}$: the set of sampled directed edges in the FORWARD algorithm.
- $\mathcal{V}(\mathcal{S})$: the nodes of the sampled edge set $\mathcal{S}$.
- $\mathcal{E}(\mathcal{V}_i)$: the set of edges in $\mathcal{E}_D$ interconnecting nodes in $\mathcal{V}_i$.
- $\mathcal{N}(\mathcal{V}_i)$: the set of nodes in $\mathcal{V}_D \backslash \mathcal{V}_i$ directly connected to nodes in $\mathcal{V}_i$.
- $(i \to j)$: a directed edge indicating flow from node i to j.

We define *current nodal value* vector (often referred to as value vector) $\mathbf{p}$, associated with the nodes of $\mathcal{V}_D$ as the unified input/output value vector across the distribution network, initialize at $\mathbf{p}_i = -d_i < 0$ for $i \in \mathcal{V}_c$ and $\mathbf{p}_i = g_i > 0$ for $i \in \mathcal{V}_g$. In our algorithmic solutions, this vector can be updated at each iteration of the algorithm, depending on the process described.

A *quasi-bipartite graph* is a graph whose vertices can be partitioned into two disjoint subsets, say U and V, such that the vast majority of edges connect vertices from U to V. Unlike strictly bipartite graphs, however, a small number of edges may exist within U or within V [15]. A 2-*core* subgraph, or 2-core, is a maximal subgraph of a graph where all vertices have a degree of at least 2. You can find the 2-core of a graph G by repeatedly removing all vertices that have a degree less than 2 in G. When you remove a vertex, its incident edges are also removed, which can reduce the degrees of its neighbors. This process is iterated until no more vertices can be removed. The remaining subgraph is the 2-core. An *articulation* node in a graph is a node which, if removed, causes the graph to be disconnected. A bi-connected graph is a graph with no articulation nodes. In a graph, a *pendent node* (also called a leaf node or end vertex) is a node (or vertex) that has a degree of exactly one, meaning it is connected to only one other node. A *pendent edge* is an edge connected to a pendent node. We let $\mathcal{G}_P$ to denote the 2-core subgraph of $\mathcal{G}_D$ and $\bar{L}$ is the total number of articulation nodes in $\mathcal{G}_P$.

A *queue* is a linear data structure in which elements are *inserted* at the rear and *retrieved* from the front (FIFO data structure: First-In, First-Out).

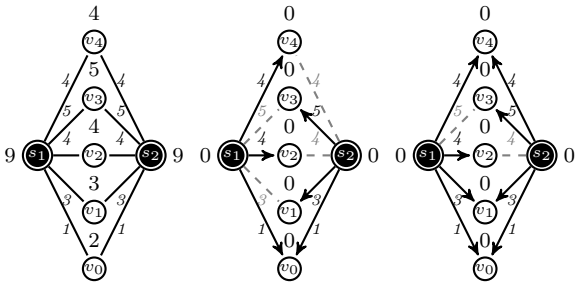
4 Hardness analysis and objective statement

Problem (1) is known to be NP-hard to solve optimally [21]. Since this paper aims to construct a feasible suboptimal solution, we next investigate the computational complexity of finding any feasible solution for Problem (1). Our analysis, formalized in Theorem 2, reveals that determining a feasible distribution configuration for problem (1) is weakly NP-complete. This classification is a particularly significant finding, as it indicates that despite its overall NP-hardness, the problem's dependence on the numerical values of its input permits the development of pseudo-polynomial time algorithms to find a feasible solution.

Theorem 2 *(Computational complexity of finding a feasible radial distribution configuration for problem* (1)*): Finding a feasible radial distribution configuration for problem* (1) *is weakly NP-complete.*

Proof. To prove the problem is in NP, we must demonstrate that a given candidate solution can be verified in polynomial time. A candidate solution consists of a set of chosen directed edges $\mathcal{S} \subseteq \mathcal{E}_D$ (directed graph $(\mathcal{V}_D, \mathcal{S})$) and their corresponding flow values $\mathbf{x}(\mathcal{S})$. Recall $N = |\mathcal{V}_D|$ and $m = |\mathcal{E}_D|$. We verify the solution in three steps with polynomial computational cost:

- *Step 1 (Check capacity constraint is respected; computational complexity $O(m)$):* Check whether the capacity constraint along very edge in $\mathcal{S}$ is satisfied.



(a) Distribution network $\mathcal{G}_D$.

(b) A feasible radial distribution.

(c) A non-radial distribution.

Fig. 4. An example of a distribution scenario described in the proof of Theorem 2. Capacities are shown as edge weights. Source nodes are shown in dark

- *Step 2 (Check flow conservation at each node; computational complexity $O(m+n)$):* Check whether flow conservation (Krichhoff's law) at each node is satisfied for the given $\mathbf{x}(\mathcal{S})$.
- *Step 3 (Check whether $(\mathcal{V}_D, \mathcal{S}) \in \mathcal{F}$; computational complexity $O(m+n)$):* Compute the undirected degree of each node in $(\mathcal{V}_D, \mathcal{S})$, i.e., sum of all the incident edges at i regardless of direction. Insert all the nodes with undirected degree of one in queue Q, which was initialized empty. Repeat until Q is empty: retrieve node i from Q. Remove the sole incident edge (i, j) of i from $\mathcal{S}$ and decrement the undirected degree of node j. If j's degree becomes 1 insert in Q. Confirm that Q become empty after $|\mathcal{S}|$ element retrieval from Q.

If all the aforementioned steps are checked out, the solution is confirmed and feasible. Since this verification steps complete in polynomial time with respect to the input size, the problem is in NP.

Next, we demonstrate weak NP-completeness by reduction from the Partition Problem, which is known to be weakly NP-complete [3]. The Partition Problem asks: given a multiset $\mathcal{A} = \{a_1, a_2, \cdots, a_n\}$ of positive integers, can $\mathcal{A}$ be divided into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that the sum of elements in $\mathcal{A}_1$ equals the sum of elements in $\mathcal{A}_2$? (This implies that both sums must be equal to $\frac{1}{2}\sum_{i=1}^{n} a_k$) [3]. We proceed as follows:

(1) Create a graph $\mathcal{G}_D$ with $n+3$ nodes: two source nodes s_1, s_2, and $n+1$ demand nodes $v_0, v_1, ..., v_n$.
(2) Add edges from both sources to all demand nodes, i.e., (s_1, v_i) and (s_2, v_i) for all $i \in \{0, 1, \cdots, n\}$.
(3) Set the demand of each node v_i to integer $d_i = a_i$, $i \in \{0, \cdots, n\}$.
(4) Set the edge capacity between node s_i, $i \in \{1, 2\}$ and demand node v_0 to $a_0/2$. For the rest of demand nodes, set the edge capacities to $\bar{x}_{s_k, v_i} = a_i$, $k \in \{1, 2\}$, $i \in \{1, \cdots, n\}$.
(5) Set the supply of each source node to $g_i = \frac{1}{2}\sum_{k=1}^{n} a_k$, $i \in \{s_1, s_2\}$.

Any feasible radial distribution network for this instance of problem (1) must include a direct edge from each source s_1 and s_2 to node v_0 each supplying $a_0/2$ to meet the demand of this node. To guarantee a feasible radial distribution, the remaining sink nodes can only be exclusively connected to one of the sources (see Fig. 4). Therefore, finding the feasible solution becomes an instance of Partition Problem. Since the Partition Problem is weakly NP-complete, problem (1) is also weakly NP-hard. As it is also in NP, it is weakly NP-complete. □

Creating a feasible solution for problem (1) involves solving a combinatorial problem whose search space grows rapidly with network dimensionality. The next result provides the maximum number of possible combinations. Specifically, we demonstrate that each feasible distribution configuration, which is a polyforest, can be augmented by adding zero-flow edges to form an underlying undirected spanning tree of $\mathcal{G}_D$, and that the total number of such spanning trees (which includes those that do not yield a feasible flow) is given by the determinant of the cofactor of the Laplacian and defines the total number of possible feasible distribution networks.

Lemma 3 *(Computational complexity of feasible solution of* (1)*): In a distribution network $\mathcal{G}_D$ with n nodes, m edges and n_g source nodes the following assertions hold about the feasible solutions of problem* (1)*: the number of feasible distribution configuration for problem* (1) *is at most det($\mathcal{L}'(\mathcal{G}_D)$). Here, det($\mathcal{L}'(\mathcal{G}_D)$) is the determinant of the co-factor of the Laplacian of the graph $\mathcal{G}_D$.*

Proof. Let $\mathcal{G}_F = (\mathcal{V}_D, \mathcal{S})$ be a feasible distribution configuration, which is a solution to Problem (1). By Definition 1, $\mathcal{G}_F$ is a polyforest that includes all nodes $\mathcal{V}_D$ and has roots at all or some of $\mathcal{V}_g$ [3]. The underlying undirected graph of $\mathcal{G}_F$, denoted $\mathcal{G}_{F,undir} = (\mathcal{V}_D, \mathcal{S}_{undir})$, is a spanning forest of $\mathcal{G}_D$ since $\mathcal{G}_F$ is acyclic and connects all nodes to the source set. If $\mathcal{G}_{F,undir}$ consists of $k > 1$ connected components, we can augment it by adding $k-1$ additional edges from the original network $\mathcal{E}_D \setminus \mathcal{S}_{undir}$ to connect these components. These additional edges must be chosen such that they do not form any cycles with the existing edges in $\mathcal{S}_{undir}$, thereby transforming $\mathcal{G}_{F,undir}$ into a single spanning tree of $\mathcal{G}_D$. For these newly added edges, we can assign a flow of zero ($x_{i,j} = 0$). Therefore, any feasible distribution configuration can be represented by selecting edges from an underlying undirected spanning tree of $\mathcal{G}_D$.

The Matrix Tree Theorem (Kirchhoff's Theorem) [2] states that for a connected undirected graph $\mathcal{G}_D$, the total number of its spanning trees is given precisely by the determinant of any cofactor of its Laplacian matrix, det($\mathcal{L}'(\mathcal{G}_D)$). Since every feasible distribution configuration can be augmented to form an underlying undirected spanning tree of $\mathcal{G}_D$, the number of feasible solutions is thus bounded and the number of feasible distribution configurations for problem (1) is at most det($\mathcal{L}'(\mathcal{G}_D)$).□

[3] While source nodes in $\mathcal{V}_g$ are considered roots of the polyforest, it is possible for them to have incoming edges if their own supply g_k is insufficient to satisfy all downstream demands they are responsible for, or if they act as relay nodes.

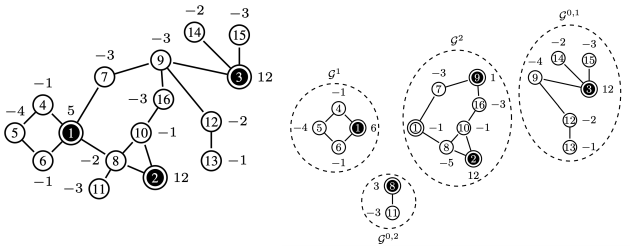
Algorithm 1 Tree-Processor

Require: Unidirected connected tree graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, value vector $\mathbf{p}$
$\mathcal{S} \leftarrow \emptyset$
while exists pendant edges in $\mathcal{E}$ **do**
  Pick a pendant edge $(i, j) \in \mathcal{E}$, let i denote the pendant node
  if $\mathbf{p}_i > 0$ and $|\mathbf{p}_i| \leq \bar{x}_{i,j}$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{(i \rightarrow j)\}$
  else if $\mathbf{p}_i = 0$ **then**
    Do nothing ▷ (i, j) is not needed for flow transfer
  else if $\mathbf{p}_i < 0$ and $|\mathbf{p}_i| \leq \bar{x}_{i,j}$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{(j \rightarrow i)\}$
  else
    Terminate the entire function, and declare that no feasible solution exists
  end if
  update $\mathbf{p}_j = \mathbf{p}_j + \mathbf{p}_i$
  $\mathcal{V} \leftarrow \mathcal{V} \backslash \{i\}$ and $\mathcal{E} \leftarrow \mathcal{E} \setminus \{(i, j)\}$
end while
return $\mathcal{G}_F(\mathcal{V}, \mathcal{S})$

A simple flow concept, starting from the pendant nodes and propagating demands/supplies inwards until no pendant edge remains, can be employed to find a feasible solution over a tree graph. This process can be completed in polynomial time, specifically $O(m + n)$, where n is the number of nodes and m is the number of edges. This approach is formalized in Algorithm 1, which guarantees either a feasible solution or a declaration of infeasibility. Furthermore, the subsequent result demonstrates that any existing solution on a tree graph is unique, and therefore, the solution generated by Algorithm 1 is also unique.

Lemma 4 *(Uniqueness of Flow and Radial Configuration in Trees): Given an undirected acyclic graph (a tree graph) where the nodes are either sink or source nodes with prespecified input and output values such that total input equals total output, the resulting flow distribution is unique. Consequently, the radial distribution configuration (polyforest) formed by these flows is also unique.*

Proof. For a connected acyclic graph with N nodes, such as a tree graph, its incidence matrix A has full column rank $(N-1)$ [15]. This implies that the null space of the system of flow conservation equations, $A\mathbf{x} = \mathbf{g} - \mathbf{d}$, is trivial. Given that the total input matches the total demand ($\sum_{i \in \mathcal{V}_g} g_i = \sum_{i \in \mathcal{V}_c} d_i$), the system is consistent.



(a) Distribution graph $\mathcal{G}_D$ (b) Partitioned subgraphs

Fig. 5. A distribution network $\mathcal{G}_D$ (plot (a)) and partitioned subgraphs with adjusted nodal value at separation nodes (plot (b)).

A consistent linear system with a trivial null space has a unique solution. Therefore, the flow vector $\mathbf{x}$ across all edges is uniquely determined. Since the flow on each edge is uniquely determined, and a radial configuration (polyforest) is formed by the directed edges carrying non-zero flow, this flow distribution uniquely defines the set of active edges and their directions. Thus, the directed radial configuration is also unique. □

Building on Lemma 3 and Lemma 4, and utilizing Algorithm 1, which establishes that every feasible distribution configuration corresponds to an underlying undirected spanning tree of $\mathcal{G}_D$, a brute-force approach to Problem (1) would involve enumerating all spanning trees of $\mathcal{G}_D$ and then checking each for feasibility. Due to the presence of capacity constraints, in the worst case, this exhaustive examination is theoretically necessary to guarantee finding a feasible solution. However, this is computationally intractable: the number of spanning trees can be exponential in n (e.g., n^{n-2} for a complete graph with n nodes), leading to an exponential computational complexity for such a brute-force method and rendering it impractical for real-world distribution networks. To circumvent this challenge, we present an algorithm in the following section that efficiently identifies a feasible solution in polynomial time $\mathcal{O}(n^2 \log n)$.

5 Graph decomposition for parallel processing

Distribution networks often contain nodes with low connectivity, creating what can be considered "pendent" radial subgraphs that are intrinsic to the original network $\mathcal{G}_D$. These subgraphs must necessarily be part of any feasible solution, as they have no alternative paths for flow. Beyond these inherent structures, distribution networks frequently possess other articulation points (cut vertices) that allow us to further partition the graph into smaller connected components.

We propose a partitioning strategy that begins by first identifying the 2-core of the distribution graph $\mathcal{G}_D$, denoted by $\mathcal{G}_P$, and its leaf components. By partitioning the 2-core further at some of its articulation nodes (will be made more clear in proceeding section), we partition the entire network $\mathcal{G}_D$ into a set of disjoint connected components, $\mathcal{G}^{0,1}, \cdots, \mathcal{G}^{0,K}, \mathcal{G}^1, \ldots, \mathcal{G}^L$, where $\mathcal{G}^{0,i}$ are connected components of $\mathcal{G}^0$, the collection of all leaf components. The rest of $\mathcal{G}^j$s are derived from partitioning the 2-core subgraph $\mathcal{G}_P$ at some of its articulation points. An example of this decomposition is illustrated in Fig. 5. To ensure that a feasible flow can be found for each subproblem, the nodal values at the separation points must be adjusted such that each component becomes individually input/output balanced. This can be achieved through an iterative or recursive process, where the flow requirements of components with only a single separating node are first resolved and their net flow propagated to the adjacent component, similar to the Tree-Processor algorithm, until all subgraphs are balanced.

The next theorem establishes the theoretical foundation for this decomposition approach by proving a fundamental equivalence between the feasibility of the original problem and the feasibility of the balanced subproblems.

Theorem 5 *(Compositional Feasibility of Radial Distributions under Graph Partitioning): Consider optimization problem* (1) *under Assumption 1. Let $\mathcal{G}_D$ be partitioned into disjoint components $\mathcal{G}^0, \mathcal{G}^1, \ldots, \mathcal{G}^L$ where $\mathcal{G}^0 = \mathcal{G}_D \backslash \mathcal{G}_P$ and $\mathcal{G}^1, \ldots, \mathcal{G}^L$ are obtained by partitioning the 2-core $\mathcal{G}_P$ at $L \leq \bar{L}$ articulation nodes. After partitioning, nodal values are adjusted at separation nodes to ensure each component $\mathcal{G}^i$, $i \in \{0, \ldots, L\}$, is individually input/output balanced. Under this setting, a feasible radial distribution network exists for the original problem* (1) *if and only if there exist feasible radial distributions for each balanced component $\mathcal{G}^\ell$, $\ell \in \{0, \ldots, L\}$.* □

To conserve space, the proof is presented in Appendix A.

Beyond feasibility, the separable structure of the quadratic cost function across disjoint edge sets ensures that optimality is also preserved under this decomposition, as formalized in the following result.

Corollary 6 *(Optimality Preservation under Graph Partitioning): Under the conditions of Theorem 5, let $(\mathcal{S}^{\ell\star}, \mathbf{x}^{\ell\star})$ be an optimal solution to problem* (1) *restricted to each balanced component $\mathcal{G}^\ell$, $\ell \in \{0, \ldots, L\}$. Then the combined solution $(\mathcal{S}^\star, \mathbf{x}^\star) = \left(\bigcup_{\ell=0}^{L} \mathcal{S}^{\ell\star}, \{\mathbf{x}^{\ell\star}\}_{\ell=0}^{L}\right)$ is an optimal solution to the original problem* (1) *on $\mathcal{G}_D$.*□

To conserve space, the proof is presented in Appendix A.

Together, Theorem 5 and Corollary 6 provide the complete theoretical justification for decomposing large-scale radial distribution problems into smaller, independent subproblems that can be solved in parallel without loss of optimality.

Algorithm 2 FORWARD

Require: Bidirectional graph $\mathcal{G}_D$, value vector $\mathbf{p}$
1: $(\mathcal{S}^0, \mathcal{V}_g, \mathcal{G}_P, \mathbf{p}) \leftarrow$ Pre-Processor$(\mathcal{G}_D, \mathbf{p})$
2: $(\mathcal{G}^1, \mathcal{V}_g^1, \mathbf{p}^1), \cdots, (\mathcal{G}^L, \mathcal{V}_g^L, \mathbf{p}^L) \leftarrow$Islander$(\mathcal{G}_P, \mathcal{V}_g, \mathbf{p})$
3: **for** each partition $\ell \in \{1, \cdots, L\}$ **do**
4: $\quad \mathbb{T}^\ell \leftarrow \emptyset$ ▷ Initializes the set of sampled polytrees
5: $\quad$ **for** $i \in \mathcal{V}_g^\ell$ **do** ▷ Initializes the sampled polytrees
6: $\quad\quad \mathcal{T}_i^\ell = \mathcal{G}(\{i\}, \{\})$
7: $\quad\quad \mathbb{T}^\ell \leftarrow \mathbb{T}^\ell \cup \mathcal{T}_i^\ell$
8: $\quad$ **end for**
9: $\quad \mathcal{S}^\ell \leftarrow \emptyset$ ▷ Initializes the sampled edges in $\mathcal{G}^\ell$
10: $\quad e^\star \leftarrow \{\}$
11: $\quad$ **while** $|\mathcal{V}(\mathcal{S}^\ell)| \neq |\mathcal{V}^\ell|$ **do**
12: $\quad\quad (\bar{\mathcal{G}}^\ell, \bar{\mathbf{p}}^\ell, \mathbb{T}^\ell) \leftarrow$Net-Concad$(\mathcal{G}^\ell, \mathbb{T}^\ell, \mathbf{p}^\ell, e^\star)$
13: $\quad\quad e^\star \leftarrow$ Sampler$(\bar{\mathcal{G}}^\ell, \mathbf{p}^\ell, \mathcal{S}^\ell)$
14: $\quad\quad \mathcal{S}^\ell \leftarrow \mathcal{S}^\ell \cup \{e^\star\}$
15: $\quad$ **end while**
16: $\quad$ **if** at least one element of $\bar{\mathbf{p}}^\ell \neq 0$ **then**
17: $\quad\quad \mathcal{S}^\ell \leftarrow$ Rewire$(\bar{\mathcal{G}}^\ell, \mathbf{p}^\ell, \mathcal{S}^\ell)$
18: $\quad$ **end if**
19: **end for**
20: $\mathcal{S} \leftarrow \bigcup_{\ell=0}^{L} \mathcal{S}^\ell$
21: **return** Radial configuration $\mathcal{G}(\mathcal{V}(\mathcal{S}), \mathcal{S})$

6 Algorithm Design for a Feasible Radial Distribution

This section introduces our proposed FORWARD algorithm, designed to construct a radial configuration within a distribution network $\mathcal{G}_D$ that yields a feasible solution for the optimal distribution problem (1). FORWARD is built on the premise of Theorem 5, which establishes that a feasible solution for the entire network can be achieved by finding feasible solutions within its decomposed parts, provided these parts are made component-wise input-output balanced by adjusting values at their separation points. FORWARD comprises five main functions, which we will briefly introduce as we outline the algorithm's structure (Algorithm 2), followed by a detailed explanation of each.

The FORWARD algorithm (Algorithm 2) takes as input the distribution network graph $\mathcal{G}_D$, its initial nodal value vector $\mathbf{p}$, and its set of source nodes $\mathcal{V}_g$. The assumption is the problem (1) is well-defined and Assumption 1 holds. The FORWARD algorithm first calls the Pre-Processor function (line 1 of Algorithm 2).

- The Pre-Processor function simplifies the initial network by iteratively processing pendant nodes and redistributing their input/output to parent nodes. This reduces problem complexity by eliminating trivial components that must necessarily be included in any feasible solution.

Next, the Islander function is invoked (line 2 of Algorithm 2).

- The Islander function partitions the 2-core subgraph $\mathcal{G}_P$ into disjoint subgraphs by splitting at articulation source nodes. This enables parallel processing and more efficient exploration of the solution space.

FORWARD then enters its parallel processing phase for each partitioned subgraph $\mathcal{G}^\ell$ to produce a feasible radial distribution configuration (lines 3-19). The algorithm initializes a set of polytrees $\mathbb{T}^\ell$ (lines 4-8), where each polytree $\mathcal{T} \in \mathbb{T}^\ell$ corresponds to a distinct source node from $\mathcal{V}_g^\ell$. These polytrees will incrementally 'grow' through edge sampling by the Sampler and Rewire functions to eventually cover the entire subgraph. During this process, polytrees may merge when connecting edges are sampled. An empty sampled edge set $\mathcal{S}^\ell$ is also initialized for each subgraph $\mathcal{G}^\ell$ (line 9).

After the aforementioned steps, FORWARD calls the Net-Concad function.

- The Net-Concad function condenses each subgraph $\mathcal{G}^\ell$ by grouping polytrees into super source nodes and unsampled sink components into super sink nodes. This creates a quasi-bipartite structure $\bar{\mathcal{G}}^\ell$ that simplifies subsequent sampling and addresses the shortsightedness of the incremental greedy process.

After this initial call to Net-Concad, FORWARD enters an iterative loop (lines 11-15). Within each iteration, every dual graph $\bar{\mathcal{G}}^\ell$ must be updated due to polytree growth. Subsequently, the Sampler function is called.

- The Sampler selects an edge to add to the evolving radial configuration based on a set of complex prioritization criteria involving weights and network flow constraints to ensure a feasible and near-optimal process.

This iterative process, driven by the repeated application of Net-Concad and Sampler, dynamically modifies the super-node structure as polytrees grow or merge. These iterations continue until an attempt is made to cover all nodes, which requires at most $|\mathcal{V}^\ell| - 1$ calls to the Sampler function.

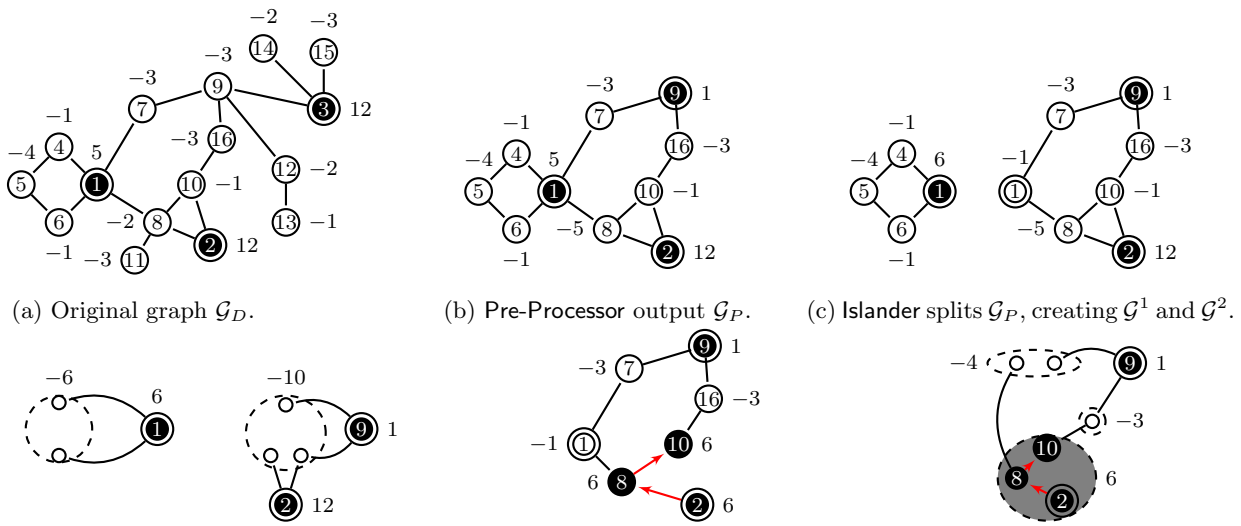
When capacity constraints exist, the greedy selection process may prevent Sampler from achieving complete node coverage. In that case, the Rewire operator (lines 16-18) is activated.

- The Rewire function identifies edges that could provide missing flow to incompletely supplied nodes and detects which sampled edges are blocking their selection. Through swapping operations, it corrects previously sampled edges toward feasibility.

6.1 Algorithm's components

6.1.1 Pre-Processor

The Pre-Processor function (Algorithm 3) takes the original distribution network graph $\mathcal{G}_D$ and the associated



(a) Original graph $\mathcal{G}_D$.

(b) Pre-Processor output $\mathcal{G}_P$.

(c) Islander splits $\mathcal{G}_P$, creating $\mathcal{G}^1$ and $\mathcal{G}^2$.

(d) Net-Concad creates $\bar{\mathcal{G}}^1, \bar{\mathcal{G}}^2$ for the sub-graphs given in plot (b).

(e) $\mathcal{G}^2$ after multiple sampling.

(f) Net-Concad output when its input graph in plot (e).

Fig. 6. Demonstration of how Pre-Processor,Islander and Net-Concad function. Filled solid nodes represent sources while others represent the sinks. The number next to the nodes show corresponding p_i. In plot (b) p_9 and p_8 are adjusted to reflect, respectfully, excess input to deliver through node 9 to $\mathcal{G}_p$ and extra demand at node 8 to supply to the removed pendent node 11, being $\mathcal{S} = \{(9 \to 12), (12 \to 13), (8 \to 11), (3 \to 14), (3 \to 15)\}$. In plot (c), after partitioning the graph, node 1 assumes different roles in each sub-graph. In plot (d) and (f) dashed circles represent super nodes. In plot (e), $\mathcal{T}_1 = \mathcal{G}(\{9\}, \{\})$ and $\mathcal{T}_2 = \mathcal{G}(\{2, 8, 10\}, (2 \to 8), (8 \to 10))$; note that here, Sampler has removed edge $(2, 10)$ to avoid cycle. Being $\mathcal{S} = \{(9 \to 12), (12 \to 13), (8 \to 11), (3 \to 14), (3 \to 15), (2 \to 8), (8 \to 10)\}$.

Algorithm 3 Pre-Processor

Require: Unidirected graph $\mathcal{G}_D$, value vector $\mathbf{p}$
1: $\mathcal{S} \leftarrow \emptyset$
2: $\mathcal{E}_P \leftarrow \mathcal{E}_D$
3: **while** exists pendant nodes **do**
4: Pick a pendent edge $(i, j) \in \mathcal{E}_P$, let i denote the pendent node
5: **if** $\mathbf{p}_i > 0$ **then**
6: $\mathcal{S} \leftarrow \mathcal{S} \cup \{(i \to j)\}$
7: **else if** $\mathbf{p}_i = 0$ **then**
8: Do nothing ▷ (i, j) is not needed for flow transfer
9: **else**
10: $\mathcal{S} \leftarrow \mathcal{S} \cup \{(j \to i)\}$
11: **end if**
12: update $\mathbf{p}_j = \mathbf{p}_j + \mathbf{p}_i$
13: $\mathcal{E}_P \leftarrow \mathcal{E}_P \setminus \{(i, j)\}$
14: **end while**
15: $\mathcal{G}_P \leftarrow \mathcal{G}(\mathcal{V}(\mathcal{E}_P), \mathcal{E}_P)$
16: Reshape $\mathbf{p}$ to include only the elements corresponding to nodes of $\mathcal{G}_P$
17: $\mathcal{V}_g = \{i \in \mathcal{V}(\mathcal{E}_p) | \mathbf{p}_i > 0\}$ ▷ Reshape $\mathcal{V}_g$ according to $\mathcal{G}_P$
18: **return** $\mathcal{S}$, $\mathcal{G}_P$, $\mathbf{p}$, $\mathcal{V}_g$

initial nodal value vector $\mathbf{p}$ as inputs. The function identifies existing radial subgraphs in $\mathcal{G}_D$ and constructs a feasible radial distribution over these subgraphs by systematically processing trivially included components.

Specifically, the Pre-Processor samples every edge connected to a pendant node (a node with degree one), such as edges $(8, 11)$, $(9, 12)$, $(12, 13)$, $(3, 14)$, $(3, 15)$, and $(9, 3)$ in Fig. 6a. For each pendant node i connected to node j, the algorithm adds the appropriately directed edge to the solution set $\mathcal{S}$ based on the sign of $\mathbf{p}_i$ (lines 5-11); Notice that if $\mathbf{p}_i = 0$, no edge is needed for flow transfer. The pendant node is then removed from the graph, and its input/output value is redistributed to its parent node (line 12). This process iterates until no pendant nodes remain in the updated graph. The result of applying the Pre-Processor function to the network in Fig. 6a is depicted in Fig. 6b. The Pre-Processor function is similar to Algorithm 1, but due to Assumption 1 that guarantees the existence of a feasible solution and Theorem 2 that establishes feasibility within radial components of $\mathcal{G}_D$, there is no need to check whether capacity constraints are satisfied during this process.

After Pre-Processor simplification, all nodes in the resulting subgraph $\mathcal{G}_P$ have a degree of at least two, making $\mathcal{G}_P$ the 2-core subgraph of $\mathcal{G}_D$. The function returns the sampled edge set $\mathcal{S}$, the 2-core subgraph $\mathcal{G}_P$, the updated nodal value vector $\mathbf{p}$, and the updated source node set $\mathcal{V}_g$ (lines 15-18).

6.1.2 *Islander*

The Islander function (Algorithm 4) takes as input the 2-core subgraph $\mathcal{G}_P$ generated by Pre-Processor, along with the corresponding source nodes $\mathcal{V}_g$ and nodal value vector $\mathbf{p}$. Note that $\mathcal{V}_g$ may differ from the original source set of $\mathcal{G}_D$ due to the Pre-Processor's redistribution of nodal values (line 17 of Algorithm 3).

Algorithm 4 Islander

Require: The 2-core subgraph $\mathcal{G}_P$, source nodes $\mathcal{V}_g$, value vector $\mathbf{p}$

1: Find nodes in $\mathcal{V}_g$ that are articulation nodes of $\mathcal{G}_P$
2: Partition $\mathcal{G}_P$ at articulation nodes creating $\mathcal{G}^\ell = \mathcal{G}(\mathcal{V}^\ell, \mathcal{E}^\ell)$, $\ell \in \{1, \cdots, L\}$
3: Define an value vector $\mathbf{p}^\ell$, which contains the elements of $\mathbf{p}$ that corresponds to nodes in $\mathcal{G}^\ell$, $\ell \in \{1, \cdots, L\}$
4: Compute and update $\mathbf{p}_i^\ell$ for each articulation point i used in partitioning $\mathcal{G}_P$ in each partition ℓ
5: $\mathcal{V}_g^\ell = \{i \in \mathcal{V}^\ell | \mathbf{p}_i^\ell > 0\}, \quad \ell \in \{1, \cdots, L\}$
6: **return** $(\mathcal{G}^1, \mathcal{V}_g^1, \mathbf{p}^1), \cdots, (\mathcal{G}^L, \mathcal{V}_g^L, \mathbf{p}^L)$

Islander partitions the the 2-core subgraph $\mathcal{G}_P$ into disjoint subgraphs $\mathcal{G}^\ell, \ell \in \{1, \ldots, L\}$, by splitting at articulation source nodes ($L \leq |\mathcal{V}_g|$, since not all source nodes are articulation nodes). The choice to partition only at source articulation nodes, rather than sink articulation nodes, reduces computational complexity while maintaining effectiveness, as sink articulations are naturally handled by the subsequent Net-Concad operation.

After partitioning, the value vector $\mathbf{p}$ of each articulation node is adjusted to balance the overall input-output in each subgraph through a simple LP problem or a process explained in Section 5. This process can change the role of articulation nodes from source to sink or vice versa within different subgraphs. For example, applying Islander to the graph in Fig. 6b (with source nodes 1, 2, and 9) creates two subgraphs as shown in Fig. 6c, where node 1 has different roles in each subgraph.

6.1.3 Net-Concad

The Net-Concad function (Algorithm 5) takes as input a graph $\mathcal{G}^\ell$ and the set of existing sampled radial polytrees $\mathbb{T}^\ell$ within $\mathcal{G}^\ell$ and a newly sampled edge $i \to j$ by the Sampler function, which will be explained next. Initially, each $\mathcal{T}_i^\ell \in \mathbb{T}^\ell$ is a distinct source node from $\mathcal{V}_g^\ell$. As new edges are sampled by the Sampler, the polytrees incrementally expand to cover the entire $\mathcal{G}^\ell$. In the process, some or all of the polytrees can merge and form larger polytrees.

Net-Concad carries two functions: one is to update the exiting polytree set after an edge is sampled by the Sampler, which is done by expanding the correspondent polytree in $\mathbb{T}^\ell$ or merging them if the edge is between to existing sampled polytrees. Note that in the absence of the capacity bound, every node within a polytree inherently can have access to the remaining positive input in the source nodes of the polytree to supply to its neighboring sink nodes (alternatively becoming into a "super sink" if a new node is added that demands more output than the polytree's current total nodal value).

The second mission of Net-Concad function is to condense each subgraph $\mathcal{G}^\ell$ into a dual graph $\bar{\mathcal{G}}^\ell$ featuring cluster nodes. This condensation is achieved by grouping

Algorithm 5 Net-Concad

Require: sub-graph $\mathcal{G}^\ell$, set of existing polytrees $\mathbb{T}^\ell$, nodal value vector $\mathbf{p}^\ell$ and newly sampled edge $i \to j$.

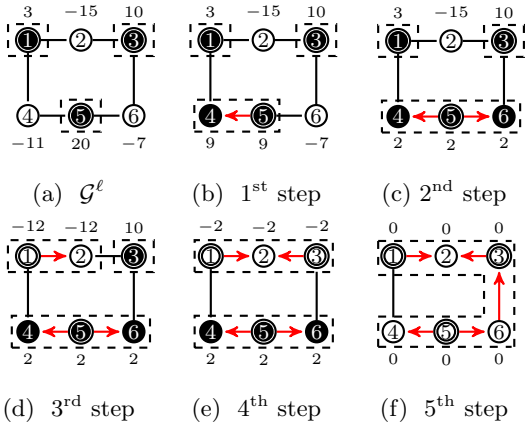
1: $\mathbb{T}^\ell \leftarrow$ Tree-Update$(\mathbb{T}^\ell, i \to j)$ ▷ See Appendix B
2: $\mathbf{p}^\ell \leftarrow$ Update$(\mathbf{p}^\ell, i \to j)$
3: Define $\mathcal{V}_s^\ell \leftarrow \{i : \forall i \in \mathcal{V}(\mathbb{T}^\ell)\}$ and $\mathcal{V}_u^\ell \leftarrow \{i : \forall i \in \mathcal{V}^\ell \setminus \mathcal{V}(\mathbb{T}^\ell)\}$
4: Define $\mathcal{G}_s^\ell \leftarrow \mathcal{G}(\mathcal{V}_s^\ell, \mathcal{E}(\mathcal{V}_s^\ell))$ and $\mathcal{G}_u^\ell \leftarrow \mathcal{G}(\mathcal{V}_u^\ell, \mathcal{E}(\mathcal{V}_u))$
5: $\bar{\mathcal{G}}_s^\ell, \bar{\mathcal{G}}_u^\ell \leftarrow$ Run Conncomp over $\mathcal{G}_s^\ell$ and $\mathcal{G}_u^\ell$
6: $\bar{\mathcal{G}}^\ell \leftarrow \mathcal{G}(\bar{\mathcal{V}}_s^\ell \cup \bar{\mathcal{V}}_u^\ell, \mathcal{E}_P^\ell \setminus \{\mathcal{E}(\mathcal{V}_s^\ell \cup \mathcal{V}_u^\ell)\})$ ▷ Re-connect sub-graphs
7: **return** $\bar{\mathcal{G}}^\ell, \mathbf{p}^\ell$

nodes within each formed polytree into a single super 'sampled' node, and by grouping each connected sink components formed after removing the super 'sampled' nodes into super un-sampled nodes, using Conncomp method described in [25] (as depicted in Figures 6d and 6f). This new dual graph $\bar{\mathcal{G}}^\ell$ is quasi-bipartite, with super 'sampled' nodes forming one partition and super 'un-sampled' nodes forming the other. Formally, we define the dual concatenated graph as $\bar{\mathcal{G}}^\ell(\bar{\mathcal{V}}^\ell, \bar{\mathcal{E}}^\ell)$, where $\bar{\mathcal{V}}^\ell$ is the node set consisted of sample and un-sampled super nodes and $\bar{\mathcal{E}}^\ell$ is the edge set where every edge $e \in \bar{\mathcal{E}}^\ell$ is a tuple (u', v', s, t), where u' and v' are supernodes in $\bar{\mathcal{V}}$, s is a subnode in the supernode represented by u' and t is a subnode in the supernode v'. $\bar{\mathcal{G}}^\ell$ is an undirected graph; when an edge (u', v', s, t) is between a super sampled node and a super un-sampled node, we let u' be the super sampled node and v' be the super un-sampled node. We define the *collective nodal value* of any *un-sampled* super node v as $\bar{\mathbf{p}}_v^\ell = \sum_{i \in \mathcal{W}_v^\ell} \mathbf{p}_i^\ell$, where $\mathcal{W}_v^\ell$ is the set of nodes in super node v. We set $\bar{\mathbf{p}}_u^\ell = 0$ for any any *sampled* super node u.

6.1.4 Sampler

The Sampler function (Algorithm 6) takes as input the graph $\bar{\mathcal{G}}^\ell$, the current nodal value vector $\mathbf{p}^\ell$ and the grown polytrees, and returns a newly sampled edge to be added to one of the polytrees. Sampler employs Weight$(\bar{\mathcal{G}}^\ell, \bar{\mathcal{G}}^\ell, \mathbf{p}^\ell, \bar{\mathbf{p}}^\ell)$ function to determine the "most probable" edge (edge with the highest weight) to include from a set of candidate edges.

The Weight function assigns a weight $w_{i,j} > 0$ to every edge $(u, v, i, j) \in \bar{\mathcal{E}}_\ell$ if only $\bar{\mathbf{p}}_u^\ell > 0$ or only $\bar{\mathbf{p}}_v^\ell > 0$, i.e., when XOR$(\bar{\mathbf{p}}_u^\ell > 0, \bar{\mathbf{p}}_v^\ell > 0) = 1$; otherwise $w_{i,j} = 0$. The weighting process assigns a positive weight to an edge if and only if it connects a super-sampled node with a positive collective nodal value to another super-node (sampled or un-sampled) with a non-positive collective nodal value. The Weight function returns all the tuples $((u, v, i, j), w_{i,j})$ with positive weight. This ensures the Sampler only considers edges where one end node consistently has a positive input to supply, thereby pushing flow from sources to sinks. These weights dynamically change after each sampling event, being defined by the specific problem context and the physical characteristics



(a) $\mathcal{G}^\ell$ (b) 1st step (c) 2nd step

(d) 3rd step (e) 4th step (f) 5th step

Fig. 7. Illustration of the sampling procedure. In b) and c), note that prioritization in Sampler avoids the flow blockage.

of the flow. Section 9 provides a concrete example of the electric power distribution Weight function.

Finally, when Sampler is called in `FORWARD` it will sample an edge by strategically using a priority queue. It first populates this queue with tuples $((u, v, i, j), w_{i,j})$ outputted from Weight. This queue is primarily sorted by descending weight, but further prioritized: highest for v being a pendent super un-sampled node or (u, v) that possesses sufficient capacity to carry the required flow. Meanwhile Weight function definition advocates for the optimality on the sampled solution, the priority queue is crucial for yielding into a feasible configuration.

Figure 7 illustrates an example case where, after repeated execution of the Sampler function, `FORWARD` results in a feasible radial distribution in the shown subgraph $\mathcal{G}^\ell$. This is an example where we do not have capacity constraints on the edges.

6.1.5 Rewire

When capacity constraints prevent Sampler from supplying all nodes, Rewire (Algorithm 7) corrects the infeasible solution through strategic edge swapping. The key insight is that unsupplied nodes can be reached through alternative paths if we reroute flow from oversupplied regions.

We define a *source-to-leaf path* $\mathcal{R}_k^i$ as a path from a source node to a leaf node within polytree $\mathcal{T}_i^\ell$. Note that $\mathcal{T}_i^\ell = \bigcup_{k=1}^r \mathcal{R}_k^i$, where r is the total number of the leaf nodes in polytree $\mathcal{T}_i^\ell$. The algorithm uses three key sets: $\mathcal{C}(\mathcal{R}, v)$ contains source-to-leaf paths that have edges connecting to unsupplied node v in the original graph $\mathcal{G}^\ell$; $\mathcal{C}(\mathcal{R}^+)$ contains *surplus source-to-leaf paths* from the already-sampled polytrees where leaf nodes have remaining supply ($\mathbf{p}^\ell > 0$); and $\mathcal{C}(\bar{\mathcal{R}})$ contains *saturated source-to-leaf paths* from the sampled polytrees where leaf nodes are not fully supplied due to capacity limits along the path.

Algorithm 6 Sampler

Require: Concatenated dual graph $\bar{\mathcal{G}}^\ell$, existing polytrees $\mathbb{T}^\ell$, nodal value vector $\mathbf{p}^\ell$

$q \leftarrow \emptyset$; $\bar{q} \leftarrow \emptyset$; $\hat{q} \leftarrow \emptyset$
$q \leftarrow$ Weight$(\bar{\mathcal{G}}^\ell, \mathbf{p}^\ell)$
Sort q in decreasing order of weights
while q is not empty **do**
  $((u, v, i, j), w_{i,j}) \leftarrow$ Retrieve(q) ▷ See Appendix B
  if v is a pendent un-sampled super node in $\bar{\mathcal{G}}^\ell$ **then**
    $e \leftarrow (i \to j)$
    Break
  else if $\mathbf{p}_i^\ell > 0$ **then**
    if $\min\{\mathbf{p}_i^\ell, x_{i,j}\} + \min\{\mathbf{p}_j^\ell, \bar{\mathbf{p}}_v^\ell\} > 0$ **then**
      Insert$((i \to j), \bar{q})$
    else
      Insert$((i \to j), \hat{q})$
    end if
  else
    if $\mathbf{p}_i^\ell + \min\{\mathbf{p}_j^\ell, \bar{x}_{i,i}\} > 0$ **then**
      Insert$((j \to i), \bar{q})$
    else
      Insert$((j \to i), \hat{q})$
    end if
  end if
end while
$e \leftarrow$ Retrieve $([\bar{q}, \hat{q}])$
return e

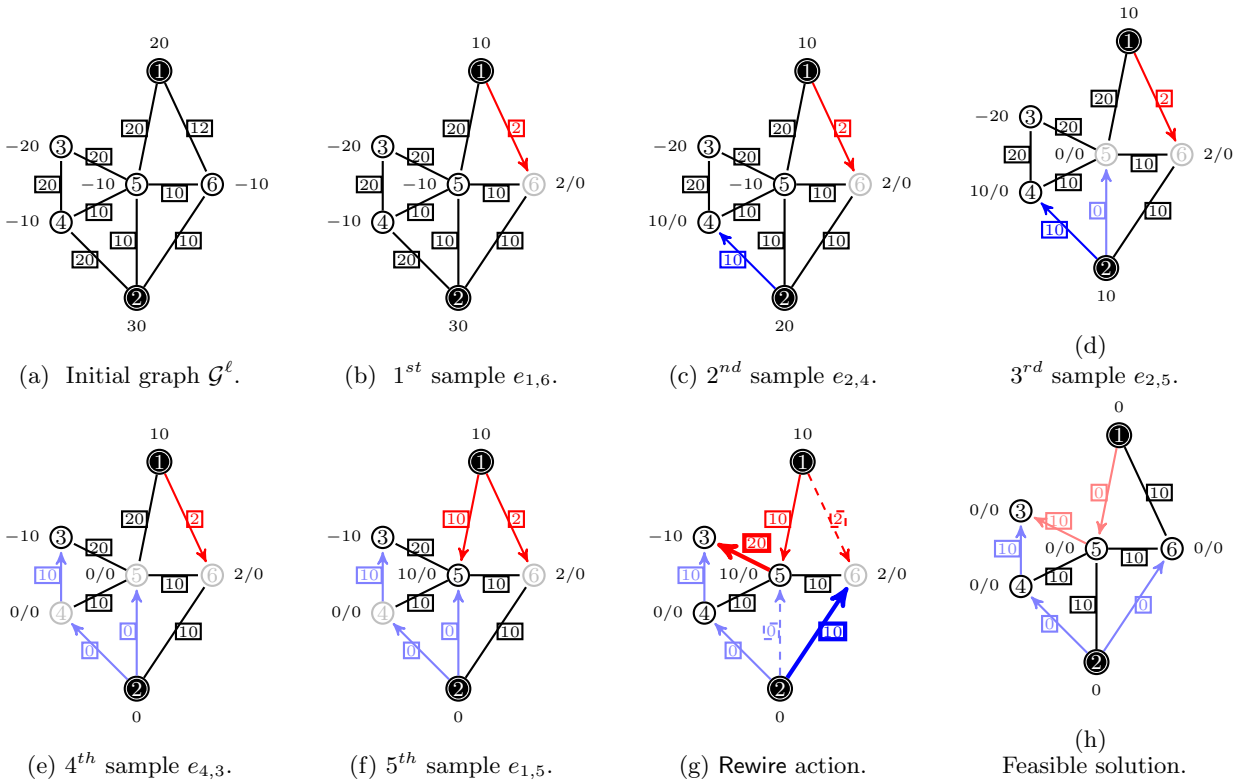
Algorithm 7 Rewire

Require: Sub-graph partition $\bar{\mathcal{G}}^\ell$, flow vector $\mathbf{p}^\ell$ and $\mathcal{S}^\ell$

1: Find $v \leftarrow \{v \in \mathcal{G}^\ell | \mathbf{p}_v^\ell < 0\}$
2: Identify $\mathcal{C}(\mathcal{R}, v)$ from $\mathcal{N}(v) \in \mathcal{S}^\ell$
3: Define $\mathcal{C}(\mathcal{R}^+)$ by finding all nodes with surplus flow in $\mathcal{S}^\ell$
4: Define $\mathcal{C}(\bar{\mathcal{R}}) \leftarrow \{\mathcal{R} \in \mathcal{S}^\ell | \mathbf{p}_{u'}^\ell < \bar{x}(u', v'), \quad \forall (u', v') \in \mathcal{R}\}$
5: **for** $\mathcal{R}_j^i \in \mathcal{C}(\mathcal{R}, v) \setminus \mathcal{C}(\bar{\mathcal{R}})$ **do**
6: Find $\{\mathcal{R}_j^{i'} \in \mathcal{C}(\mathcal{R}^+) | \exists (u', v') \in \mathcal{G}^\ell\}$ where u' is leaf of $\mathcal{R}_j^{i'}$ and $v' \in \mathcal{T}_i^\ell$.
7: Find $\hat{e}_{u',v'}$ where $u' \in \mathcal{R}_j^i$ and $v' \notin \mathcal{R}_j^i$ and $v' \in \mathcal{T}_i^\ell$
8: **if** $\exists \mathcal{R}_j^{i'}$ **then**
9: Delete $\hat{e}_{u',v'}$
10: Define $\mathcal{R}_j^i \leftarrow \mathcal{R}_j^i \cup \{v\}$, $\mathcal{R}_j^i \leftarrow \mathcal{R}_j^i \setminus \{v'\}$, $\mathcal{R}_j^{i'} \leftarrow \mathcal{R}_j^i \cup \{v'\}$
11: *break*
12: **end if**
13: **end for**
14: **return** $\mathcal{S}^\ell$

Rewire operates through a systematic swap mechanism to redistribute flow from oversupplied regions to unsupplied nodes. The algorithm works in three main phases:

Phase 1: Identify rewiring opportunities. The algorithm first locates unsupplied nodes v where $\mathbf{p}_v^\ell < 0$ (line 1) and identifies source-to-leaf paths that could potentially supply these nodes through non-sampled edges in the original graph, specifically $\mathcal{C}(\mathcal{R}, v) \setminus \mathcal{C}(\bar{\mathcal{R}})$ (lines 2-4). These represent source-to-leaf paths with sufficient capacity to reach the unsupplied nodes if properly connected. Simultaneously, the algorithm identifies surplus



(a) Initial graph $\mathcal{G}^\ell$. (b) 1^{st} sample $e_{1,6}$. (c) 2^{nd} sample $e_{2,4}$. (d) 3^{rd} sample $e_{2,5}$.

(e) 4^{th} sample $e_{4,3}$. (f) 5^{th} sample $e_{1,5}$. (g) Rewire action. (h) Feasible solution.

Fig. 8. An example demonstrating the application of the Rewire function to resolve an infeasible solution generated by Sampler alone. Input and output values at each node in the original graph $\mathcal{G}^\ell$ are shown as numbers next to the nodes. Edge capacity values are displayed in boxed weights. After sampling, the notation x/y next to each sink node indicates: x represents the excess supply available at that node, and y represents the demand at the node. When an edge is sampled to supply downstream nodes, the capacity values are adjusted based on the remaining flow that can pass through the edge, to be the excess supply at the terminal node of the sampled edge.

source-to-leaf paths $\mathcal{C}(\mathcal{R}^+)$ whose leaf nodes have remaining supply that can be redistributed.

Phase 2: Execute strategic swaps. For each viable source-to-leaf path in $\mathcal{C}(\mathcal{R}, v)\backslash\mathcal{C}(\bar{\mathcal{R}})$, the algorithm seeks surplus source-to-leaf paths $\mathcal{C}(\mathcal{R}^+)$ whose leaf nodes can donate flow (lines 5-6). When such a surplus source-to-leaf path is found, Rewire performs a swap operation: it removes a blocking edge $\hat{e}_{u',v'}$ from the current polytree structure (line 9) and establishes new connections that redirect the surplus flow to supply the previously unsupplied node v (line 10). This swap effectively transfers the surplus leaf node from one polytree to another, creating a new supply path to the unsupplied node.

Phase 3: Maintain radiality. The edge deletion and reconnection process ensures that the resulting network maintains its radial (tree-like) structure by avoiding cycles while establishing new supply paths to previously unsupplied nodes (lines 9-11). The algorithm manages these topological changes to preserve the polytree properties essential for the overall solution framework.

The key insight is that Rewire leverages the flexibility of polytree structures to redistribute existing supply capacity rather than requiring additional generation resources. By strategically swapping connections between surplus and deficit regions, the algorithm transforms an infeasible solution into a feasible one, as illustrated in Fig. 8. This approach is particularly effective because it maintains the computational efficiency of working within the sampled polytree structure while resolving capacity-induced infeasibilities through localized network modifications.

7 Feasibility Analysis of FORWARD Generated Radial Distribution Networks

This section demonstrates that the FORWARD algorithm is guaranteed to generate feasible radial distribution configurations for optimization problem (1). By Theorem 5, we established that feasibility of the original problem is equivalent to finding feasible solutions in each balanced component $\mathcal{G}^\ell$. Therefore, our analysis focuses on proving that FORWARD generates feasible radial configurations within each subgraph $\mathcal{G}^\ell$.

7.1 Feasibility Without Capacity Constraints

We begin by analyzing the case where capacity constraints are not binding, establishing the fundamental feasibility properties of FORWARD.

Lemma 7 *(Edge Direction Preservation in Sampled Polytrees): Given an edge e sampled by* Sampler *to merge two distinct polytrees, it will not reverse the directionality of previously sampled edges.*

Proof. Assume for contradiction that merging polytrees $\mathcal{T}_i^\ell$ (with aggregate nodal value $\mathbf{p}_i^\ell > 0$) and $\mathcal{T}_j^\ell$ (with $\mathbf{p}_j^\ell < 0$) via edge e causes a previously established flow direction within $\mathcal{T}_j^\ell$ to reverse. For flow reversal to occur within $\mathcal{T}_j^\ell$, there must have been an internal flow path directed towards a sink node u, which upon connection via e is forced to redirect away from u. This would only happen if the merged polytree has net positive flow that redirects existing internal flows of $\mathcal{T}_j^\ell$. However, the Sampler's priority condition requires $\mathbf{p}_i^\ell + \mathbf{p}_j^\ell \geq 0$ for edge selection. This condition ensures that the merged polytree maintains sufficient supply to meet existing demands without forcing flow reversals. The quasi-bipartite property of $\bar{\mathcal{G}}^\ell$ ensures edges connect supply-side components to demand-side components, with flow naturally moving from positive to negative nodal values. Within the tree structure of polytrees, flow directions are uniquely determined by source-sink relationships once demands are fixed (as established in Lemma 4). The Sampler's aggregate nodal value check and the tree-like nature of polytrees ensure that previously directed edges maintain their flow direction when polytrees are merged. □

Lemma 8 *(Properties of FORWARD's Output): After FORWARD execution on each subgraph $\mathcal{G}^\ell(\mathcal{V}^\ell, \mathcal{E}^\ell)$, the resulting graph $\mathcal{G}(\mathcal{V}(\mathcal{S}^\ell), \mathcal{S}^\ell)$ is a radial configuration covering all nodes ($\mathcal{V}(\mathcal{S}^\ell) = \mathcal{V}^\ell$). If the initial input-output flow is balanced in $\mathcal{G}^\ell$, then every node's demand is met with $\mathbf{p}_i^\ell = \mathbf{0}$ for all $i \in \mathcal{V}^\ell$.*

Proof. The proof demonstrates that FORWARD consistently generates a radial configuration, ensures complete node coverage, and guarantees flow balance when the initial subgraph is balanced.

Radial Configuration (Polyforest Property): The process initiates with each source node forming a trivial polytree. The FORWARD algorithm's core loop iteratively adds edges via Sampler. The Sampler operates on the dual graph $\bar{\mathcal{G}}^\ell$ constructed by Net-Concad, where existing polytrees are condensed into super sampled nodes and un-sampled components into super un-sampled nodes. The Sampler's edge selection ensures that each chosen edge (s, t) either (a) connects an existing polytree to an un-sampled component or (b) merges two distinct existing polytrees. In both cases, a single edge is added between previously disconnected components, strictly preventing cycle formation and ensuring $\mathcal{G}(\mathcal{V}(\mathcal{S}^\ell), \mathcal{S}^\ell)$ remains a polyforest.

Complete Node Coverage and Flow Balance: We prove that the generated polyforest completely covers the graph ($\mathcal{V}(\mathcal{S}^\ell) = \mathcal{V}^\ell$) and delivers inputs from sources to meet all demands within a finite number of steps.

Upon Islander application, each subgraph $\mathcal{G}^\ell$ has one or more nodes with positive nodal value, collectively balancing the total demand across $\mathcal{G}^\ell$. Since each $\mathcal{G}^\ell$ has minimum degree 2, the dual graph $\bar{\mathcal{G}}^\ell$ constructed by Net-Concad is consistently connected. This connectivity ensures every un-sampled super node in $\bar{\mathcal{G}}^\ell$ connects to at least one super sampled node.

At each step, as long as not all nodes are covered, there exists at least one super sampled node with positive nodal value (since total flow in $\mathcal{G}^\ell$ is balanced and unmet sink demands require corresponding excess supply in sampled polytrees). The Sampler prioritizes edges based on its priority list, systematically extending polytrees by connecting them to un-sampled nodes or merging distinct polytrees.

The Weight function assigns positive weight $w_{i,j} > 0$ only to edges where precisely one connected super-node has positive nodal value ($\mathsf{XOR}(\mathbf{p}_i^\ell > 0, \mathbf{p}_j^\ell > 0) = 1$), guiding Sampler's choices. By prioritizing connections that can be fully supplied ($\mathbf{p}_i^\ell + \mathbf{p}_j^\ell > 0$), this process effectively distributes positive nodal values from sources towards un-sampled super nodes until flow is balanced.

This iterative process terminates because $|\mathcal{V}^\ell|$ is finite. Each step either adds a new node to the covered set or merges existing polytrees, ensuring monotonic progress towards full coverage. The process stops when all sampled polytrees have zero nodal value, which occurs only when all nodes are covered ($\mathcal{V}(\mathcal{S}^\ell) = \mathcal{V}^\ell$) and all demands are met, resulting in $\mathbf{p}_i^\ell = \mathbf{0}$ for all $i \in \mathcal{V}^\ell$. □

Theorem 9 *(FORWARD Generates Feasible Radial Configurations (Uncapacitated Case)): The FORWARD algorithm (Algorithm 5) without capacity constraints is guaranteed to construct feasible radial configurations for optimization problem* (1).

Proof. A feasible configuration for problem (1) requires radial structure (polyforest), complete node coverage, and balanced flow distribution. The proof analyzes the sequential operations of FORWARD and properties of its outputs with regards to these attributes.

(1) **Pre-Processor Operation:** The Pre-Processor separates radial subgraphs from the 2-core subgraph

($\mathcal{G}_P$) at articulation points, balancing nodal values to ensure input-output balance within each component. By Lemma 4, the polyforest created over these separated radial configurations is unique and meets all demands within their respective components.

(2) **Islander Operation:** The Islander function partitions the 2-core subgraph $\mathcal{G}_P$ at articulation points with positive nodal value, adjusting nodal values to ensure each newly created subgraph $\mathcal{G}^\ell$ is input-output balanced.

(3) **FORWARD Operation on each $\mathcal{G}^\ell$:** For each balanced subgraph $\mathcal{G}^\ell$, Lemma 8 proves that FORWARD ensures: (a) radial configuration construction, (b) complete node coverage, and (c) flow balance with all demands met.

(4) **Overall Feasibility:** Each component processed by Pre-Processor and each subgraph $\mathcal{G}^\ell$ processed by FORWARD yields a balanced radial configuration. These configurations are joined at their respective articulation points, resulting in a complete radial and flow-balanced configuration for the entire $\mathcal{G}_D$ that satisfies all feasibility criteria for optimization problem (1). □

7.2 Feasibility With Capacity Constraints

When capacity constraints are present, FORWARD may initially produce infeasible solutions where some nodes cannot be fully supplied due to capacity bottlenecks along their source-to-leaf paths. As described earlier, the Rewire function addresses this by redistributing flow through strategic edge swapping between surplus and saturated source-to-leaf paths.

Lemma 10 *(Rewire Identifies and Resolves Infeasibility): Given any capacity-infeasible solution from FORWARD, Rewire can identify blocking edges and redistribute flow to achieve feasibility when a feasible solution exists.*

Proof. Given an unsupplied node v (where $\mathbf{p}_v^\ell < 0$), by the flow balance enforced by Islander, there must exist a source-to-leaf path $\mathcal{R}_j^{i'}$ from polytree $\mathcal{T}_{i'}^\ell$ that can supply the remaining demand through a non-capacity-constrained route:

$$\exists \mathcal{R}_j^{i'} \in \mathcal{C}(\mathcal{R}, v) \backslash \mathcal{C}(\bar{\mathcal{R}}) \text{ with leaf node } u \text{ such that } (u, v) \in \mathcal{G}^\ell$$

By flow balance, there also exists a polytree $\mathcal{T}_i^\ell$ with surplus flow, meaning $\mathcal{T}_i^\ell \cap \mathcal{C}(\mathcal{R}^+) \neq \emptyset$. This provides a source-to-leaf path $\mathcal{R}_j^i \in \mathcal{C}(\mathcal{R}^+)$ with leaf node v' that has excess supply. Rewire performs the following swap: (1) connects the surplus leaf v' to polytree $\mathcal{T}_{i'}^\ell$, (2) redirects the non-constrained path to supply node v, and (3) removes any blocking edge $\hat{e}_{u',v'}$ that would create a cycle. This maintains radiality while achieving feasibility. □

Theorem 11 *(FORWARD with Rewire Generates Feasible Solutions): The complete FORWARD algorithm including Rewire is guaranteed to construct feasible radial configurations for optimization problem* (1) *when feasible solutions exist.*

Proof. The proof follows from Theorem 9 and Lemma 10. When capacity constraints are not binding, Theorem 9 guarantees feasibility. When capacity constraints create infeasibilities, Lemma 10 demonstrates that Rewire can identify and resolve these infeasibilities through strategic flow redistribution while maintaining the radial structure. Since Rewire operates under the assumption that a feasible solution exists (Assumption 1), the complete algorithm is guaranteed to find such a solution. □

8 Complexity

The computational complexity of FORWARD is determined by the sequential execution of its sub-processes. For each sub-process, we analyze their complexity as follows:

- Pre-Processor: $\mathcal{O}(n + m)$ as it processes all pendant nodes and their incident edges, potentially iterating through the entire graph structure.
- Islander: $\mathcal{O}(n + m)$ for finding articulation points and partitioning the graph using standard graph algorithms.
- Net-Concad: $\mathcal{O}(m)$ per call for condensing the graph into super-nodes using connected component algorithms.
- Sampler: $\mathcal{O}(m \log m)$ per call due to priority queue operations over candidate edges. This function is called at most $n - 1$ times to construct the spanning forest.
- Rewire: $\mathcal{O}(n+m)$ in the worst case, where it may need to examine all source-to-leaf paths and perform edge swapping operations.

The overall complexity is dominated by the iterative sampling process: $\mathcal{O}((n+m)+(n+m)+(n-1)\cdot(m+m\log m)+(n+m))$, which simplifies to $\mathcal{O}(nm \log m)$.

However, this analysis represents the worst-case scenario. In practice, the complexity is significantly reduced due to several factors: *Graph Sparsity:* Distribution networks typically exhibit sparse, small-world properties where $m = \mathcal{O}(n)$ [14]. Under this assumption, the complexity reduces to $\mathcal{O}(n^2 \log n)$; *Problem Decomposition:* After Pre-Processor, the remaining 2-core subgraph $\mathcal{G}_P$ contains significantly fewer than n nodes. Subsequently, Islander partitions this reduced graph into even smaller

subgraphs $\mathcal{G}^\ell$, each processed independently and potentially in parallel; *Early Termination:* The Sampler often terminates before examining all m edges due to the priority-based selection mechanism and the quasi-bipartite structure of the dual graph $\bar{\mathcal{G}}^\ell$.

Therefore, while the theoretical worst-case complexity is $\mathcal{O}(n^2 \log n)$ for sparse networks, the practical performance is substantially better due to the algorithm's structure-aware design and the inherent properties of distribution networks.

9 Demonstration Example

To demonstrate the computational benefits of `FORWARD`, we evaluate its performance on optimal power distribution problems where generating feasible radial configurations is a computationally challenging and recurring task. We consider six different power distribution networks: three IEEE standard test systems (IEEE 13 with 2 sources, IEEE 18 with 2 sources, IEEE 33 with 3 sources) and three small-world networks (WS 120 with 10 sources, WS 240 with 10 sources, and WS 400 with 20 sources).

9.1 Problem Formulation for Power Networks

The IEEE graphs have been preprocessed to single-phase DC network structures. As established in the literature [12], single-phase direct current (DC) simplifications provide effective approximations to three-phase alternating current (AC) flow problems, allowing the optimal power flow problem to be cast in the form of Problem (1). The small-world networks are generated using the Watts-Strogatz mechanism [30] to create realistic network topologies with small-world properties.

In power distribution networks, the parameters of Problem (1) have specific physical interpretations:

- d_i represents power demand at load nodes (sinks)
- g_i represents power generation at generator nodes (sources)
- $x_{i,j}$ represents power flow through transmission line (i, j).

The power loss across edge $e_{i,j}$ is computed as $R_{i,j}\left(\frac{x_{i,j}}{v_j}\right)^2$, where v_j is voltage at node j and is a fix value. The voltage variation across the distribution network often is neglected. Using per unit (p.u.) representation we assume that $v_j = 1$ p.u. for all nodes $i \in \mathcal{V}_D$ [22]. Therefore, in problem (1) we have $C_{i,j} = R_{i,j}$.

For this problem we design the sampling weights based on how power flows across a distribution line. Following Ohm's law, the power flow $x_{i,j}$ through edge $(i \to j)$ depends on the total power demand $d_{i\to j}$ of all loads supplied through that edge and the voltage v_j at node j. The power loss cost through edge $(i \to j)$ is:

$$f_{(i\to j)} = R_{i,j} \cdot \left(\frac{d_{i\to j}}{v_j}\right)^2$$

Since all edges within a source-to-leaf path $\mathcal{R}_k^i$ carry the same flow, we define a rolling-demand estimator:

$$\hat{f}(\mathcal{R}_k^i) = \sum\nolimits_{\kappa\in\mathcal{R}_k^i} R_{\kappa,\kappa+1} \cdot d_\kappa$$

The sampling weight for the Sampler function is then defined as:

$$w_{i,j} = \frac{\mathbf{p}_i}{R_{i,j} \cdot d_j^2 + \hat{f}(\mathcal{R}_k^i)} \tag{2}$$

where $\mathbf{p}_i$ is the current nodal value representing remaining power at node i, which decreases at each iteration as power demand is allocated. The weights $w_{i,j}$ are dynamically updated and normalized at each iteration across all candidate edges.

9.2 Experimental Setup, Results and Analysis

We implemented our numerical experiments using the PowerDistributionModel (PMD) framework [11] from Los Alamos National Laboratory, coded in Julia. For comparison, we use Knitro from Artelys to solve the MINLP formulation of Problem (1). All experiments were conducted on a MacBook Air with an M3 chip and 24 GB of RAM. The source codes for this simulation study is available at [27].

The simulation results are presented in Table 1 and Figures 9 and 10. Power loss values are reported in kilowatts (kW). In the time column, * indicates processes manually terminated after excessive computation time, and "TL" (Time Limit) indicates cases where no solution was found within 3 hours. The results demonstrate several key advantages of `FORWARD`:

Computational Efficiency: Knitro was only able to return solutions for small-sized networks, with CPU times significantly exceeding those of `FORWARD`. Commercial solvers like Knitro rely on heuristics for warm-start initialization within the feasible domain. As network size increases, these heuristics struggle to find proper initialization points within polynomial time due to the exponential growth of the combinatorial space, often resulting in prohibitive computation times or complete failure to find initial feasible points.

Scalability: In contrast, `FORWARD` successfully constructed feasible radial configurations for all test cases with remarkably low computation times, even for networks with 400 nodes. The algorithm's polynomial-time

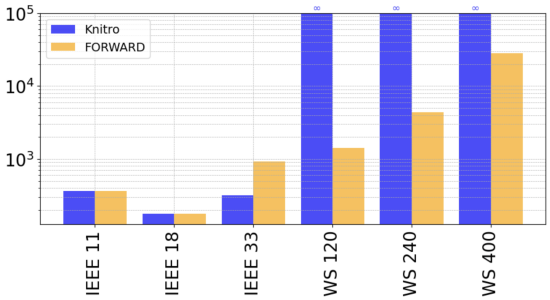


Fig. 9. Power loss comparison between Knitro and FORWARD.

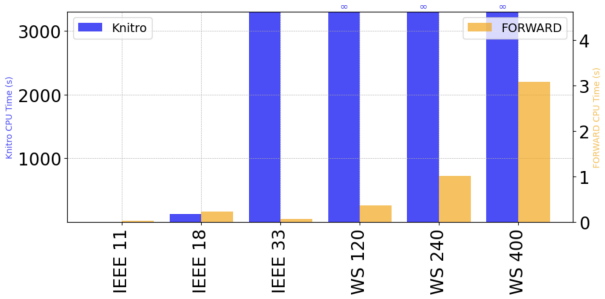


Fig. 10. CPU time comparison between Knitro and FORWARD.

Table 1
Performance comparison between Knitro and FORWARD.

	Knitro		FORWARD	
Network	Power Loss (kW)	CPU Time (s)	Power Loss (kW)	CPU Time (s)
IEEE 13	360.183	4.189	360.183	0.033
IEEE 18	175.821	123.416	175.821	0.229
IEEE 33	318.568	3,345.67*	919.783	0.066
WS 120	TL	TL	1,428.72	0.361
WS 240	TL	TL	4,393.17	1.016
WS 400	TL	TL	28,345.7	3.090

complexity enables it to handle large-scale networks that are intractable for exact MINLP solvers.

Solution Quality: For cases where both methods found solutions (IEEE 13 and IEEE 18), FORWARD achieved identical power loss values, demonstrating optimal performance on these instances. For the IEEE 33 network, while Knitro was terminated before completion, FORWARD provided a feasible solution with reasonable power loss in a fraction of the time. The promising solution quality can be attributed to the physically-aware weight design in Equation (2) and the systematic sampling mechanism that considers both electrical properties and network topology.

Practical Impact: These results highlight FORWARD's practical value for real-world power distribution planning, where computational tractability is essential for operational decision-making. The algorithm's ability to handle networks of realistic size (400+ nodes) within seconds makes it suitable for online optimization and real-time grid management applications.

10 Conclusions

This paper presented FORWARD, a polynomial-time algorithm for constructing feasible radial configurations in multi-source distribution networks. We addressed the optimal radial reconfiguration problem (1) and proved that feasibility determination was weakly NP-complete, making exact methods computationally intractable for large networks. Our approach consisted of three main contributions. First, we developed a graph decomposition strategy (Theorem 5) that enabled parallel processing while preserving feasibility and optimality. Second, we designed a five-function algorithmic framework where Net-Concad addressed greedy shortsightedness through dual graph condensation, and Rewire handled capacity constraints via strategic edge swapping. We provided rigorous theoretical analysis proving feasibility guarantees for both uncapacitated and capacitated cases. Third, numerical evaluation on six networks (up to 400 nodes) demonstrated FORWARD's superior computational efficiency compared to commercial MINLP solvers. While traditional methods required hours or failed entirely, FORWARD achieved solutions in seconds with optimal or near-optimal quality. The algorithm's success stemmed from leveraging radial network structure and incorporating physically-motivated weight functions rather than relying on generic branch-and-bound approaches. The polynomial-time complexity made FORWARD suitable for real-time distribution network management applications. Despite being designed for quadratic cost models, the algorithm successfully handled complex physics constraints in power system reconfiguration, establishing its effectiveness as an initialization strategy for iterative optimization solvers.

Future work included formally characterizing the optimality gap, extending to nonlinear constraint models, and investigating applicability to broader network optimization problems. The promising results suggested that our algorithmic principles could impact telecommunications, supply chain, and transportation planning beyond

radial distribution networks.

References

[1] M. E. Baran and F. F. Wu. Network reconfiguration in distribution systems for loss reduction and load balancing. *IEEE Transactions on Power delivery*, 4(2):1401–1407, 1989.

[2] Béla Bollobás. *Modern Graph Theory*. Springer New York, 1998.

[3] Christian Borgs, Jennifer Chayes, and Boris Pittel. Phase transition and finite-size scaling for the integer partitioning problem. *Random Structures amp; Algorithms*, 19(3–4):247–288, October 2001.

[4] Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *SIAM Journal on Computing*, 46(4):1280–1303, January 2017.

[5] L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. P. Gutenberg, and S. Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *Annual Symposium on Foundations of Computer Science*, pages 612–623, 2022.

[6] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs, 2010.

[7] Andrew Clark. A submodular optimization approach to the metric traveling salesman problem with neighborhoods. In *IEEE Conference on Decision and Control*, pages 3383–3390, Nice, France, 2019.

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition, 2001.

[9] Sarineh Hacopian Dolatabadi, Maedeh Ghorbanian, Pierluigi Siano, and Nikos D. Hatziargyriou. An enhanced ieee 33 bus benchmark test system for distribution system studies. *IEEE Transactions on Power Systems*, 36(3):2565–2572, 2021.

[10] Peter G. Doyle and J. Laurie Snell. Random walks and electric networks, 2000.

[11] David M. Fobes, Sander Claeys, Frederik Geth, and Carleton Coffrin. Powermodelsdistribution.jl: An open-source framework for exploring distribution power flow formulations. *Electric Power Systems Research*, 189:106664, 2020.

[12] David M. Fobes, Harsha Nagarajan, and Russell Bent. Optimal microgrid networking for maximal load delivery in phase unbalanced distribution grids: A declarative modeling approach. *IEEE Transactions on Smart Grid*, 14(3):1682–1691, 2023.

[13] Guillaume Guex. Interpolating between random walks and optimal transportation routes: Flow with multiple sources and targets. *Physica A: Statistical Mechanics and its Applications*, 450:264–277, May 2016.

[14] Bálint Hartmann and Viktória Sugár. Searching for small-world and scale-free behaviour in long-term historical data of a real-world power grid, 2020. arXiv:2010.09315.

[15] Daniel Hsu. Coms 3251: Connectivity in graphs. *Columbia*, page 1–10, October 2022.

[16] R. A. Jabr, R. Singh, and B. C. Pal. Minimum loss network reconfiguration using mixed-integer convex programming. *IEEE Transactions on Power Systems*, 27(2):1106–1115, 2012.

[17] David R. Karger. Minimum cuts in near-linear time, 1998.

[18] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, page 217–226, USA, 2014.

[19] A. Khodabakhsh, G. Yang, S. Basu, E. Nikolova, M. C. Caramanis, T. Lianeas, and E. Pountourakis. A submodular approach for electricity distribution network reconfiguration. In *Hawaii Int. Conf. on System Sciences*, pages 2717–2726, 2018.

[20] Russell Lyons and Yuval Peres. *Minimal Spanning Forests*, page 388–409. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2017.

[21] A. Merlin and H. Back. Search for a minimal-loss operating spanning tree configuration in an urban power distribution system. In *Power System Computation Conference*, volume 5, pages 1–18, Cambridge, UK, 1975.

[22] Q. Peng and S. H. Low. Optimal branch exchange for feeder reconfiguration in distribution networks. In *IEEE Conference on Decision and Control*, pages 2960–2965, Florence, Italy, 2013.

[23] Ezequiel C. Pereira, Carlos H. N. R. Barbosa, and João A. Vasconcelos. Distribution network reconfiguration using iterative branch exchange and clustering technique. *Energies*, 16(5):2395, March 2023.

[24] Nikolaos V. Sahinidis. Mixed-integer nonlinear programming 2018. *Optimization and Engineering*, 20(2):301–306, April 2019.

[25] Robert E. Tarjan and Uri Zwick. Finding strong components using depth-first search, 2022.

[26] Stephen R. Tate. Proof of correctness for prim's algorithm, handout 11, November 2016. Class Handout, CSC 330: Advanced Data Structures.

[27] Joan Vendrell. Forward. `https://github.com/joanvendrell/FORWARD`, 2025. Accessed: 2025-10-08.

[28] Joan Vendrell, Russell Bent, and Solmaz Kia. Forward: Feasibility oriented random-walk inspired algorithm for radial reconfiguration in distribution networks, 2024.

[29] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[30] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.

[31] R. Y. Yamamoto, T. Pinto, R. Romero, and L. H. Macedo. Specialized tabu search algorithm applied to the reconfiguration of radial distribution systems. *International Journal of Electrical Power & Energy Systems*, 162:110258, 2024.

[32] A. F. Zobaa and A. Vaccaro. *Computational intelligence applications in smart grids: enabling methodologies for proactive and self-organizing power systems*. World Scientific, 2014.

A Proofs

Proof. [Proof of Theorem 5] Let $\mathcal{G}^\ell = (\mathcal{V}^\ell, \mathcal{E}^\ell)$ denote the ℓ-th component. For any separation node v, let

$\mathcal{C}(v) = \{\ell | v \in \mathcal{V}^\ell\}$ denote the set of components containing v.

($\Rightarrow$) Suppose there exists a feasible solution $(\mathcal{S}, \mathbf{x})$ for problem (1) on $\mathcal{G}_D$.

For each component $\mathcal{G}^\ell$ with separation nodes $\mathcal{V}^\ell_{sep} = \{v_1, v_2, \ldots, v_k\} \subset \mathcal{V}^\ell$, we construct the adjusted nodal values as follows: For each separation node $v_i \in \mathcal{V}^\ell_{sep}$, Kirchhoff's law in the original graph gives:

$$\sum_{u \in \mathcal{V}^\ell \setminus \{v_i\}} (x_{u,v_i} - x_{v_i,u}) + \sum_{\ell' \neq \ell} \sum_{u \in \mathcal{V}^{\ell'}} (x_{u,v_i} - x_{v_i,u}) = g_{v_i} - d_{v_i}.$$

Define:

$$f^\ell_{v_i} = \sum_{u \in \mathcal{V}^\ell \setminus \{v_i\}} (x_{u,v_i} - x_{v_i,u}) \quad \text{(net flow into } v_i \text{ from within } \mathcal{G}^\ell\text{)}$$
$$f^{\text{ext}}_{v_i} = \sum_{\ell' \neq \ell} \sum_{u \in \mathcal{V}^{\ell'}} (x_{u,v_i} - x_{v_i,u}) \quad \text{(net external flow into } v_i\text{)}.$$

Then $f^\ell_{v_i} + f^{ext}_{v_i} = g_{v_i} - d_{v_i}$ for each $i \in \{1, 2, \ldots, k\}$. Define the adjusted nodal values:

$$p^\ell_{v_i} = g_{v_i} - d_{v_i} - f^{ext}_{v_i} = f^\ell_{v_i} \quad \text{for separation nodes}$$
$$p^\ell_u = g_u - d_u \quad \text{for internal nodes } u \in \mathcal{V}^\ell \setminus \mathcal{V}^\ell_{sep}.$$

To verify that component $\mathcal{G}^\ell$ is balanced, we compute:

$$\sum\nolimits_{u \in \mathcal{V}^\ell} p^\ell_u = \sum\nolimits_{u \in \mathcal{V}^\ell \setminus \mathcal{V}^\ell_{sep}} (g_u - d_u) + \sum\nolimits_{i=1}^k f^\ell_{v_i}.$$

By Kirchhoff's law applied to the subgraph $\mathcal{G}^\ell$, since internal nodes can only exchange flow with separation nodes (no external connections):

$$\sum\nolimits_{u \in \mathcal{V}^\ell \setminus \mathcal{V}^\ell_{sep}} (g_u - d_u) + \sum\nolimits_{i=1}^k f^\ell_{v_i} = 0.$$

Therefore, $\sum_{u \in \mathcal{V}^\ell} p^\ell_u = 0$, confirming that $\mathcal{G}^\ell$ is balanced. When we restrict the original flow $\mathbf{x}$ to edges within $\mathcal{G}^\ell$, obtaining $\mathbf{x}^\ell$, the flow conservation is automatically satisfied:

- For internal node $u \in \mathcal{V}^\ell \setminus \mathcal{V}^\ell_{sep}$:

$$\sum\nolimits_{w|(u,w) \in \mathcal{E}^\ell} (x^\ell_{w,u} - x^\ell_{u,w}) = g_u - d_u = p^\ell_u$$

- For separation node $v_i \in \mathcal{V}^\ell_{sep}$:

$$\sum\nolimits_{w|(v_i,w) \in \mathcal{E}^\ell} (x^\ell_{w,v_i} - x^\ell_{v_i,w}) = f^\ell_{v_i} = p^\ell_{v_i}$$

Hence, $A(\mathcal{S}^\ell)\mathbf{x}^\ell = \mathbf{p}^\ell$ is satisfied for each component $\mathcal{G}^\ell$.
($\Leftarrow$) Suppose there exist feasible solutions $(\mathcal{S}^\ell, \mathbf{x}^\ell)$ for each balanced component $\mathcal{G}^\ell$ satisfying $A(\mathcal{S}^\ell)\mathbf{x}^\ell = \mathbf{p}^\ell$.

Construct the global solution as $\mathcal{S} = \bigcup_{\ell=0}^L \mathcal{S}^\ell$ and $\mathbf{x}$ as the concatenation of all $\mathbf{x}^\ell$.

The capacity constraints are satisfied since they hold for each component individually and the edge sets are disjoint. The union of disjoint polyforests rooted at source nodes remains a polyforest, preserving the radial property.

For flow conservation at any separation node v shared by multiple components, we have:

$$\sum\nolimits_{\ell \in \mathcal{C}(v)} \left(\sum\nolimits_{u \in \mathcal{V}^\ell \setminus \{v\}} (x^\ell_{u,v} - x^\ell_{v,u}) \right) = \sum\nolimits_{\ell \in \mathcal{C}(v)} p^\ell_v.$$

By construction of the adjusted values:

$$\sum\nolimits_{\ell \in \mathcal{C}(v)} p^\ell_v = \sum\nolimits_{\ell \in \mathcal{C}(v)} (g_v - d_v - f^{ext,\ell}_v) = g_v - d_v$$

where the external flow terms cancel out since $\sum_{\ell \in \mathcal{C}(v)} f^{ext,\ell}_v = 0$ (what flows out of one component flows into another at separation node v). For internal nodes (non-separation), flow conservation is trivially satisfied since their nodal values remain unchanged. Therefore, Kirchhoff's law is satisfied at every node in the reconstructed graph, completing the proof. □

Proof. [Proof of Corollary 6] The proof follows from the separability of the quadratic cost function across disjoint edge sets and the equivalence established in Theorem 5. Since the components $\mathcal{G}^\ell$ have disjoint edge sets $\mathcal{E}^\ell$, the total cost of any feasible solution $(\mathcal{S}, \mathbf{x})$ on $\mathcal{G}_D$ can be decomposed as:

$$\sum\nolimits_{(i,j) \in \mathcal{S}} C_{i,j} \cdot x^2_{i,j} = \sum\nolimits_{\ell=0}^L \sum\nolimits_{(i,j) \in \mathcal{S}^\ell} C_{i,j} \cdot (x^\ell_{i,j})^2,$$

where $\mathcal{S}^\ell = \mathcal{S} \cap \mathcal{E}^\ell$ and $x^\ell_{i,j}$ is the flow on edge (i,j) within component $\mathcal{G}^\ell$. By Theorem 5, there is a bijective correspondence between feasible solutions on $\mathcal{G}_D$ and collections of feasible solutions on the balanced components $\{\mathcal{G}^\ell\}_{\ell=0}^L$. Therefore:

$$\min_{(\mathcal{S},\mathbf{x}) \text{ feasible on } \mathcal{G}_D} \sum\nolimits_{(i,j) \in \mathcal{S}} C_{i,j} \cdot x^2_{i,j} =$$

$$\sum\nolimits_{\ell=0}^L \min_{(\mathcal{S}^\ell,\mathbf{x}^\ell) \text{ feasible on } \mathcal{G}^\ell} \sum\nolimits_{(i,j) \in \mathcal{S}^\ell} C_{i,j} \cdot (x^\ell_{i,j})^2$$

Since $(\mathcal{S}^{\ell,*}, \mathbf{x}^{\ell,*})$ is optimal for component $\mathcal{G}^\ell$, the combined solution achieves this minimum cost. By Theorem 5, this combined solution is feasible for the original problem, and by the cost separability, it is optimal. □

B Sub-routines

In this appendix we introduce with greater detail internal sub-routines in `FORWARD` mentioned in Section 6.

Algorithm 8 Tree-Update

Require: Existing polytree set $\mathbb{T}^\ell$ and newly sampled edge $i \to j$.

for $\mathcal{T} \in \mathbb{T}^\ell$ **do**
  if $i \in \mathcal{V}(\mathcal{T})$ **then**
    $\mathbb{T}^\ell \leftarrow \mathbb{T}^\ell \backslash \mathcal{T}$
    $\mathcal{T} \leftarrow \mathcal{T} \cup (\{i, j\}, i \to j)$ ▷ New edge added to polytree
    for $\mathcal{T}' \in \mathbb{T}^\ell \backslash \mathcal{T}$ **do** ▷ Merging polytrees if needed
      if $j \in \mathcal{V}(\mathcal{T}')$ **then**
        $\mathbb{T}^\ell \leftarrow \mathbb{T}^\ell \backslash \mathcal{T}'$
        $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}'$
      end if
    end for
    Break
  end if
end for
return $\mathbb{T}^\ell$

Algorithm 9 Conncomp [25]

Require: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Ensure: Set of connected components $\mathbb{C}$

$\mathbb{C} \leftarrow \emptyset$
visited[v] $\leftarrow$ **false** for all $v \in \mathcal{V}$
for each $v \in \mathcal{V}$ **do**
  if not visited[v] **then**
    $\mathcal{C} \leftarrow \emptyset$
    DFS(v, $\mathcal{G}$, visited, $\mathcal{C}$)
    $\mathbb{C} \leftarrow \mathbb{C} \cup \{\mathcal{C}\}$
  end if
end for
return $\mathbb{C}$
function DFS(v, $\mathcal{G}$, visited, $\mathcal{C}$)
  visited[v] $\leftarrow$ **true**
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{v\}$
  for each $u \in \mathcal{N}(v)$ **do**
    if not visited[u] **then**
      DFS(u, $\mathcal{G}$, visited, $\mathcal{C}$)
    end if
  end for
end function

Algorithm 10 Retrieve

Require: Existing queue $q = \{x_0, x_1, \cdots, x_n\}$.

$q \leftarrow q \setminus \{x_0\}$
return x_0