

An Analysis of the Backward-Forward Sweep with Counter-Monotonic Functions

Bachelor's Thesis of

Moritz Maas

At the Department of Informatics
Institute of Theoretical Informatics (ITI)

Reviewer: T.T.-Prof. Dr. Thomas Bläsius
Second reviewer: Dr. rer. nat. Torsten Ueckerdt
Advisor: Wendy Yi

16.05.2026 – 16.09.2026

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

Karlsruhe, 16.09.2026

.....

.....
(Moritz Maas)

Abstract

The *Backward-Forward Sweep (BFS)* is an iterative algorithm to solve power flow problems in radial distribution networks. When applied to electrical AC power flow, the BFS iteratively computes bus voltages that control the flow of power through the transmission lines between them.

This thesis studies the BFS as general algorithmic framework for solving flow problems in rooted trees and directed acyclic graphs. We abstract from the physical interpretation of the BFS and focus on its algorithmic properties, which are applicable to a wide range of flow problems in networks.

We begin by formalizing this framework for rooted trees, define the Flow Problem, and analyze the convergence behavior of the BFS for flow models with a monotonicity property, which we call *counter-monotonicity*. Beyond trees, the thesis extends the framework to directed acyclic graphs on which the problem becomes more complex. To address this, we reduce the Flow Problem on directed acyclic graphs to the already solvable Flow Problem on rooted trees. For this, we present an algorithm to solve the Flow Problem on directed acyclic graphs with treewidth 2 and use the BFS as a subroutine.

Zusammenfassung

Der *Backward-Forward Sweep (BFS)* ist ein iterativer Algorithmus zur Lösung von Lastflussproblemen in radialen elektrischen Verteilungsnetzen. Angewandt auf Lastflussprobleme im Wechselstrom, berechnet der BFS iterativ die Spannungen an den Netzbussen, welche die Lastflüsse in den Leitungen bestimmen.

Diese Arbeit untersucht den BFS als ein generisches Framework zur Lösung von Lastflussproblemen auf gewurzelten Bäumen und gerichteten azyklischen Graphen. Wir abstrahieren von der physikalischen Interpretation des BFS und konzentrieren uns auf die algorithmischen Eigenschaften, die auf eine Vielzahl von Lastflussproblemen anwendbar sind.

Wir beginnen mit einer formalen Definition des Frameworks auf gewurzelten Bäumen, definieren das Flow Problem und analysieren die Konvergenz des BFS für Flussmodelle mit einer Monotonieeigenschaft, welche wir *Gegenmonotonie* oder *counter-monotonicity* nennen. Über Bäume hinaus erweitern wir das Framework auf gerichtete azyklische Graphen, für die Lastflussprobleme schwerer zu lösen sind. Um diese Schwierigkeit zu überwinden, reduzieren wir das Flow Problem auf gerichteten azyklischen Graphen auf das Flow Problem auf gewurzelten Bäumen, welches wir bereits lösen können. Dafür präsentieren wir einen neuen Algorithmus, der das Flow Problem auf gerichteten azyklischen Graphen mit Baumweite 2 löst, indem er den BFS als Subroutine verwendet.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Related Work	2
1.3	Outline	3
2	Preliminaries	5
2.1	Graphs	5
2.2	Functions	5
2.3	Distributions	6
3	Flow Trees	7
3.1	Flow	7
3.2	Flow Problem	8
4	Backward-Forward Sweep	9
4.1	Backward Sweep	10
4.2	Forward Sweep	10
4.3	Backward-Forward Sweep	11
4.4	Convergence	14
4.4.1	Epsilon Convergence	16
5	Potential-Independent Backward Functions	17
5.1	Lossless Flow Functions	18
6	Counter-Monotonic Backward/Forward Functions	19
6.1	Modified Power Flow	22
6.2	Palm Trees	27
7	Non-Tree Graphs	31
7.1	Routing	31
7.2	Split Graph	32
7.3	Routed Backward-Forward Sweep	33
7.4	Disjoint Flow DAGs	36
7.4.1	Routing Search Algorithm	38
7.5	Nested Disjoint Flow DAGs	41
7.5.1	Nested Backward-Forward Sweep	43
7.5.2	Nested Routing Search	45
8	Evaluation	47
8.1	Generating Feasible Instances	47
8.1.1	Top-Down-Generation	47
8.1.2	Epsilon-Top-Down-Generation	48

8.1.3	Bottom-Up-Generation	50
8.2	Palm Trees	51
8.3	Routing Search	53
8.4	Nested Backward-Forward Sweep	54
9	Conclusion	57
9.1	Future Work	57
	Bibliography	59

1 Introduction

1.1 Motivation

Many problems in network optimization and electrical engineering require the distribution of power through a network while satisfying local demands. The flow of power through a transmission line of a network is directly controlled by the voltage potentials of the incident buses. The problem lies in finding the correct voltage potentials for each bus such that the power demands of all buses are satisfied. Such power flow problems are well-studied in electrical engineering, but often do not allow a direct computation of the solution voltage potentials.

The *Backward-Forward Sweep (BFS)* was introduced as an iterative algorithm intended for solving the AC-OPF problem on radial distribution networks [SHSL88]. When the BFS is applied to a radial network, it computes the voltage potentials of all buses in two repeated steps. First, the *Backward Sweep* computes the power flows through the transmission lines based on the current voltage potentials. Then, the *Forward Sweep* computes new voltage potentials based on the computed power flows. This process is repeated until the voltage potentials converge to a solution.

Related work established that the feasibility of AC power flows is strongly $\mathcal{NP}$ -hard [BV19]. This greatly limits the efficiency of power flow analysis in radial distribution systems. Despite this, power flow solution methods show practical success on many instances, and the BFS is widely used in practice. Therefore, it is of interest to study the BFS in a more general setting and to understand its convergence behavior for different flow models.

Most research on the BFS focuses on its application to electrical power flow problems, and therefore only considers the physical interpretation of the algorithm, where the voltage potentials and power flows are determined by the laws of physics. Further, these studies often only consider its performance on very specific and restricted instances, such as the IEEE radial distribution test feeders [Ker91], which only contain at most 123 buses. Our goal is to abstract from this and formalize a general algorithmic framework for the BFS that is applicable to a wide range of flow problems in networks. We can model these electrical networks as directed graphs, where the buses are represented by vertices with associated voltage potentials and the transmission lines by edges with associated impedances. Because most distribution networks are radial, this often results in trees where the root is the only generating source of power. Although the BFS is originally designed for these radial networks, we want to analyze it in a more general setting and study its behavior for different flow models and network topologies. The focus lies on flow models with a monotonicity property, we call *counter-monotonicity*, which is a property that is satisfied by many practically relevant flow models and guarantees some beneficial properties. We show that the BFS converges to a solution for flow models with counter-monotonic backward functions and extend the BFS framework to directed acyclic graphs. Trying to apply the BFS to more general graphs, we find that it can be used as a subroutine to solve the flow problem on directed acyclic graphs with treewidth 2.

1.2 Related Work

Power flow problems have been studied since before the invention of digital computers, going back to the early 20th century. Early digital approaches used iterative elimination and impedance matrix formulations to solve the nonlinear equations describing the network [WH56 | VG61 | BCHP63]. These methods repeatedly update the unknown bus voltages until the power flow equations are satisfied. Their computational cost and convergence behavior depend strongly on the network size, the initial voltages, and the conditioning of the corresponding system of equations.

An alternative approach uses Newton's method [TH67]. It linearizes the power flow equations around the current voltages and computes a correction by solving a system involving the Jacobian matrix of the equations. Newton's method is known to converge quadratically near a regular solution. However, this local guarantee does not imply global convergence. Poor initial values, singularities in the Jacobian, or heavily loaded networks can cause divergence [FN23 | DC13]. Moreover, repeatedly constructing and factorizing the Jacobian can be expensive for large networks. Several variants have therefore been developed, including modified Newton methods for radial distribution systems [GR99 | DVE09]. These methods exploit the structure of radial networks to reduce computational effort while retaining much of the robustness of the Newton approach [ZC97].

The Fast Decoupled Loadflow method reduces the cost of Newton iterations by neglecting selected couplings between active and reactive power as well as voltage magnitude and angle [SA74]. This produces constant, or infrequently updated, coefficient matrices and consequently makes each iteration considerably cheaper than a full Newton iteration step. The approximation can provide high computational speed on transmission networks, but its accuracy and convergence depend on the assumptions used in the decoupling. A theoretical study of its convergence shows that these assumptions and the network parameters play an important role in determining whether the algorithm converges [Wu77]. Fast decoupled formulations have also been adapted to unbalanced radial distribution systems [ZC95].

The Backward-Forward Sweep is an alternative iterative method designed particularly for radial distribution networks. It exploits the tree structure of such networks instead of repeatedly solving a global Jacobian system. The method was introduced by Shirmohammadi, Hong, Semlyen and Luo as an iterative power flow method for weakly meshed and transmission networks [SHSL88], and has since become one of the standard approaches for radial distribution system analysis.

Many variants of the BFS have been proposed. They differ in the choice of state variables, the representation of transmission line losses, and the treatment of single- and multiphase systems. For example, direct solution techniques for distribution systems exploit the radial topology to eliminate intermediate variables and compute the solution efficiently [GB91]. Other works develop simplified radial network formulations [DKK95], three-phase power flow methods [Che+91 | CS95], and robust three-phase algorithms intended to improve numerical reliability for practical distribution systems [TWJ99]. For weakly meshed networks, compensation methods combine a tree-based sweep with additional corrections for the remaining cycles [LS90].

The convergence of the BFS depends on the loading level, line impedances, initial voltages, and the particular flow model. Analytical studies have established convergence conditions for backward-forward methods under suitable assumptions [BCCN00 | FZL25]. Comparative studies also report that the BFS is faster and less computationally demanding than general purpose

methods on radial networks, while Newton-type methods may converge in fewer iterations when they converge successfully [EH08 | MMH21 | dPdB18]. Recent surveys summarize these trade-offs and the adaptations required for modern distribution networks [AMS26].

As stated above, Bienstock and Verma established that the feasibility of AC power flows is strongly $\mathcal{NP}$ -hard [BV19]. This difficulty is not removed by restricting the network to be radial. AC-feasibility remains $\mathcal{NP}$ -hard on tree networks [LGV16].

1.3 Outline

Chapter 2 introduces the notation and definitions used throughout the thesis. This includes graphs, functions, and distributions.

Chapter 3 defines flow trees and the Flow Problem, which is the problem that the BFS solves.

Chapter 4 formally defines the Backward-Forward Sweep. It also introduces convergence properties of the BFS and proves that the BFS finds a solution whenever it converges.

Chapter 5 considers potential-independent backward functions. We show that the BFS finds a solution after one iteration.

Chapter 6 introduces the concept of counter-monotonic functions, which are the focus of this thesis. We establish monotonicity properties of the vertex potentials and total demands that are used to prove convergence results. We show that the Modified Power Flow functions are counter-monotonic and take a closer look at feasibility of palm tree instances.

Chapter 7 extends the Flow Problem to directed acyclic graphs. We introduce routing functions, split graphs and the Routed Backward-Forward Sweep. Finally, we present an algorithm to solve the Flow Problem on certain directed acyclic graphs, which we call nested disjoints.

Chapter 8 evaluates the theoretical results of this thesis. We provide instance generation methods and evaluate the convergence of the BFS on palm trees. We also evaluate the proposed algorithm for solving the Flow Problem on nested disjoints.

2 Preliminaries

We introduce the necessary definitions and notations for this thesis.

2.1 Graphs

A directed graph G is a pair (V, E) , where V is a set of vertices and $E \subseteq V \times V$ is a set of directed edges. A subgraph $G' = (V', E')$ of a directed graph $G = (V, E)$ is a directed graph such that $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$. If G' is a subgraph of G , we also say that G *contains* G' and write $G' \subseteq G$.

A *path* in graph G is a sequence of vertices $(v_1, v_2, \dots, v_k)$ with $k \geq 1$ such that $(v_i, v_{i+1}) \in E$ for all $1 \leq i < k$. The length of a path is the number of edges it contains, which is $k - 1$ for a path of k vertices. Note that there is always a path of length 0 from a vertex to itself. Two paths are *disjoint* if they do not share any edges.

A directed graph G is a *directed acyclic graph* or *DAG* if it contains no cycles, i.e., there is no path $(v_1, v_2, \dots, v_k)$ such that $v_1 = v_k$ and $k > 1$. For a DAG we define the following concepts. If there exists an edge $(u, v) \in E$, we call u the *parent* of v and v the *child* of u . The set of all parents of a vertex v is denoted by $p(v) := \{u \in V : (u, v) \in E\}$. The set of all children of a vertex u is denoted by $c(u) := \{v \in V : (u, v) \in E\}$. The *ancestors* of a vertex v are all vertices $u \neq v$ that have a path to v . The *descendants* of a vertex u are all vertices $v \neq u$ where a path exists from u to v . A *sink* in a DAG is a vertex v with no children, i.e., $c(v) = \emptyset$. A DAG G is *rooted* in a vertex r if there is a path from r to every other vertex in G . We call r the *root* of the directed graph. Note that the root vertex r is unique and has no parents.

A DAG G rooted in r is a *directed tree* if every non-root vertex has exactly one parent, i.e., $|p(v)| = 1$ for all $v \in V \setminus \{r\}$. This implies that there is a unique path from the root r to every other vertex in G and that the number of edges in G is $|E| = |V| - 1$. Sink vertices in a tree are called *leaves*.

Given a DAG $G = (V, E)$, we define the *induced subgraph* G_v of a vertex v as the subgraph containing v , its descendants and all edges connecting them, that is $G_v = (V', E')$ where $V' = \{w \in V : \text{there is a path from } v \text{ to } w\}$ and $E' = \{(u, w) \in E : u, w \in V'\}$. Note that G_v is also a DAG rooted in v and that G_v is a tree if G is a tree.

Figure 2.1 shows an overview of the concepts defined in this section.

2.2 Functions

A function $f : A \rightarrow B$ is a relation between two sets A and B such that for every element $a \in A$, there exists a unique element $b \in B$ such that $(a, b) \in f$. We denote this relationship as $f(a) = b$.

Given two functions $f : A \rightarrow B$ and $g : A \rightarrow B$ with a relation $\sim \subseteq B \times B$, we define that $f \sim g$, if $f(a) \sim g(a)$ for all $a \in A$. When using $\leq$ as the relation $\sim$ we compare two functions f and g pointwise, that is f is less than or equal to g at every point of the domain A .

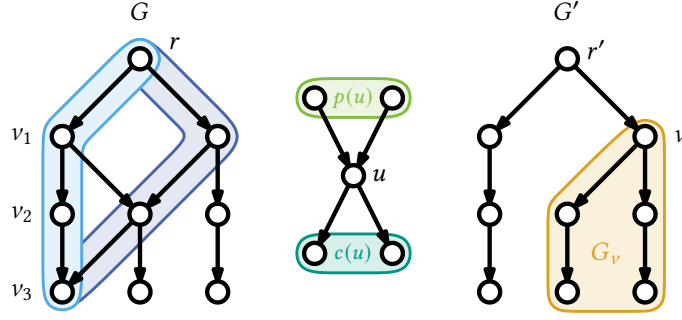


Figure 2.1: On the left, a DAG G rooted in r containing a path (r, v_1, v_2, v_3) and another disjoint path from r to v_3 . In the center, an example of the set of parents $p(u)$ and children $c(u)$ of a vertex u . On the right, a directed tree G' rooted in r' . In G' the induced subtree G_v of a vertex v is highlighted.

2.3 Distributions

For an interval $[a, b] \subseteq \mathbb{R}$, we define the *uniform distribution*, denoted by $\mathcal{U}([a, b])$, as the distribution where every value $x \in [a, b]$ is equally likely to occur, and every value $x \notin [a, b]$ has probability 0 of occurring. If a value x is drawn from a uniform distribution $\mathcal{U}([a, b])$, we denote this by $x \sim \mathcal{U}([a, b])$.

3 Flow Trees

As the foundation of our work, we consider radial distribution networks. These networks can be represented as directed trees, where the root vertex represents the power generating source and the other vertices represent the consumers of power. In this chapter, we introduce the theoretical framework behind radial distribution networks and define the power distribution problem we want to solve.

Let $G = (V, E)$ be a directed tree rooted in r and $n \in \mathbb{N}$ be a positive integer. We assign a *demand* to every vertex $S : V \rightarrow \mathbb{R}^n$ and write $S_x := S(x)$ for $x \in V$. If $S_x = 3$ for some vertex x , we interpret this as x demanding three units of flow. The root vertex r has no demand, i.e., $S_r = 0$ and should be thought of as the supplier of flow for the other vertices. We also assign a parameter to each edge $w : E \rightarrow \mathbb{R}^n$ and write $w_e := w(e)$ for $e \in E$. The parameter of an edge e can be used to model its capacity or cost of sending flow through e , for example. To fulfill the vertex demands, we assign a *potential* to each vertex $\phi : V \rightarrow \mathbb{R}^n$. The vertex potentials dictate how much flow is sent from a vertex to its children and how much flow is received from its parent. We write $\phi_x := \phi(x)$ for $x \in V$. A *flow tree* is a directed graph with demands, edge parameters and a given root potential ϕ_r . We denote such a flow tree by $G' = (V, E, \phi_r, S, w)$ and say it has dimension n .

3.1 Flow

To describe the distribution of flow through the network, we must distinguish between the flow sent from a vertex to its children and the flow received at a vertex from its parent. For each edge $e = (u, v) \in E$, we define the flow sent from u to v as $S_{u \rightarrow v}$ and the flow received at v from u as $S_{v \leftarrow u}$. Note that $S_{u \rightarrow v}$ is positive if u sends flow to v , and $S_{v \leftarrow u}$ is positive if v receives flow from u . Therefore, the excess flow at v is the incoming flow minus the outgoing flows and thus calculated as

$$S_{v \leftarrow u} - \sum_{x \in c(v)} S_{v \rightarrow x}.$$

To model the flow for each edge, we define two functions that determine the outgoing and incoming flow based on the potentials of the parent and child vertices and the parameter of the edge between them. The outgoing flow $S_{u \rightarrow v}$ is determined by the *send function*

$$f_p : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n, \quad f_p(\phi_u, \phi_v, w_e) = S_{u \rightarrow v}$$

which takes the potentials of the parent and child vertices and the parameter of the edge between them as input and outputs the flow sent from the parent to the child. The incoming flow $S_{v \leftarrow u}$ is determined by the *receive function*

$$f_c : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n, \quad f_c(\phi_u, \phi_v, w_e) = S_{v \leftarrow u}$$

which takes the same inputs as the send function but outputs the flow received at the child from its parent. We demand that the functions f_p and f_c are well-defined and continuous for all input potentials $\phi_u, \phi_v \in \mathbb{R}^n$. Together, the functions f_p and f_c are called the *flow functions* or the *flow model*. The definitions of this section are illustrated in Figure 3.1.

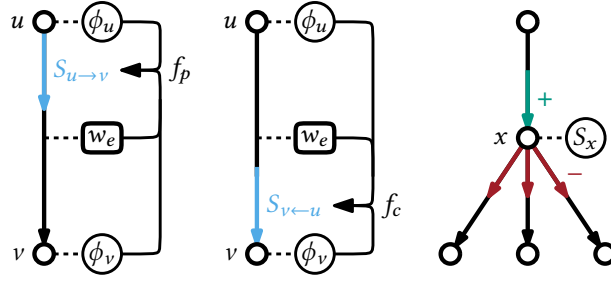


Figure 3.1: The left and center figure illustrate the calculation of outgoing and incoming flow $S_{v \leftarrow u}$ and $S_{u \rightarrow v}$ using the flow functions f_p, f_c on an edge $e = (u, v)$. The right figure shows the excess flow as the difference between the incoming and outgoing flows for a vertex x . Dashed lines indicate an associated value, e.g., the vertex potentials or edge parameter.

3.2 Flow Problem

When the flow functions f_p, f_c and potentials ϕ are given, we can compute the flow for each edge and thus the excess flow at each vertex. Our goal is to find vertex potentials such that the excess flow at each vertex satisfies its demand exactly. So, given a flow tree $G = (V, E, \phi_r, S, w)$ and flow functions f_p, f_c , the *Flow Problem* is to find vertex potentials ϕ such that for every non-root vertex $v \in V \setminus \{r\}$ with parent u the equation

$$S_v = S_{v \leftarrow u} - \sum_{x \in c(v)} S_{v \rightarrow x}$$

holds and the root potential equals ϕ_r . By using the definitions of the send and receive functions, we can rewrite this equation as

$$S_v = f_c(\phi_u, \phi_v, w_e) - \sum_{x \in c(v)} f_p(\phi_v, \phi_x, w_{(v,x)})$$

to get a formulation of the Flow Problem in terms of the flow functions f_p, f_c and the vertex potentials ϕ .

Vertex potentials ϕ that satisfy the equation above for all non-root vertices $v \in V \setminus \{r\}$ are a *solution* to the Flow Problem. Every flow tree $G = (V, E, \phi_r, S, w)$ with flow functions f_p, f_c admits an *instance* of the Flow Problem. If an instance of the Flow Problem has a solution, we say that the instance is *feasible*, otherwise it is *infeasible*.

4 Backward-Forward Sweep

To solve the Flow Problem defined in Section 3.2 for a flow tree $G = (V, E, \phi_r, S, w)$, we redefine the problem in a way that allows us to compute the vertex potentials ϕ iteratively, since most flow functions f_p, f_c do not allow us to compute the vertex potentials ϕ directly. For this, we initialize our vertex potentials with some initial values. Then, we iteratively estimate the required incoming flow $S_{v \leftarrow u}$ at every vertex v using our current vertex potentials. With this estimate, we update the vertex potentials ϕ and repeat this process.

We formalize this process by rearranging the equation from the Flow Problem and get the equivalent equation

$$S_{v \leftarrow u} = S_v + \sum_{x \in c(v)} S_{v \rightarrow x}$$

which we try to satisfy for every non-root vertex $v \in V \setminus \{r\}$ with parent u . We define the right-hand side of this equation as the *total demand* $\hat{S}_v$ of a vertex v , i.e.,

$$\hat{S} : V \rightarrow \mathbb{R}^n, \quad v \mapsto S_v + \sum_{x \in c(v)} S_{v \rightarrow x}.$$

As a shorthand, we write $\hat{S}_v := \hat{S}(v)$ for $v \in V$. We can see that the total vertex demand $\hat{S}_v$ is the initial vertex demand S_v plus the flow sent to its children. Since that flow is dependent on the current vertex potentials ϕ , we cannot compute the correct total vertex demands $\hat{S}$ of a solution directly without knowing the correct vertex potentials. But we can rewrite the equation above as

$$f_c(\phi_u, \phi_v, w_e) = \hat{S}_v$$

and solve for ϕ_v to get the *forward function* fw .

$$fw(\phi_u, \hat{S}_v, w_e) = \phi_v$$

Intuitively, the forward function fw computes the vertex potential ϕ_v needed to satisfy the total demand $\hat{S}_v$ based on the vertex potential ϕ_u of its parent and the edge parameter w_e . As a result, the forward function fw and the receive function f_c are inverses in the second argument, that is the equations

$$f_c(\phi_u, fw(\phi_u, \hat{S}_v, w_e), w_e) = \hat{S}_v, \quad fw(\phi_u, f_c(\phi_u, \phi_v, w_e), w_e) = \phi_v$$

must hold for every edge $e = (u, v) \in E$.

To calculate the total vertex demands, we need to compute the flow sent to the children of a vertex u . To calculate this, we define the *backward function* as

$$bw(\phi_u, \hat{S}_v, w_e) = f_p(\phi_u, fw(\phi_u, \hat{S}_v, w_e), w_e)$$

which computes the flow sent from u to its child v based on the parents potential ϕ_u , the child total vertex demand $\hat{S}_v$ and the edge parameter w_e . We demand that the functions fw and bw are continuous for all inputs $\phi_u, \hat{S}_v \in \mathbb{R}^n$. Since the edge parameter w_e is fixed for every edge $e \in E$, often has no effect on our results and is clear from context, we will often omit it as a function argument and write $fw(\phi_u, \hat{S}_v)$ and $bw(\phi_u, \hat{S}_v)$. An illustration of the backward and forward functions is given in Figure 4.1.

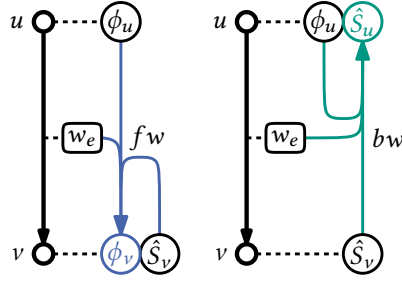


Figure 4.1: An illustration of the backward and forward functions. The left figure shows how the forward function fw computes the vertex potential ϕ_v based on the parent potential ϕ_u and the total vertex demand $\hat{S}_v$. The right figure shows how the backward function bw computes the flow sent from u to its child v based on the parent potential ϕ_u and the child's total vertex demand $\hat{S}_v$. Dashed lines indicate an associated value, e.g., the vertex potentials or edge parameter.

4.1 Backward Sweep

In the *Backward Sweep*, we want to compute the total vertex demands $\hat{S}$ for all vertices by using the backward function bw . Because the backward function bw depends on the total vertex demand of the children, we start with the leaf vertices and move upwards to the root in a bottom-up manner. For every vertex u , we compute $\hat{S}_u$ by adding its initial demand S_u to the flow sent to its children, which is computed using the backward function bw . This results in the formula

$$\hat{S}_u = S_u + \sum_{v \in c(u)} bw(\phi_u, \hat{S}_v, w_{(u,v)})$$

for every vertex u . We can see that we estimate the total vertex demand $\hat{S}_u$ as the initial vertex demand S_u plus the flow sent to its children, which is dependent on the current vertex potential ϕ_u and the total vertex demands of its children $\hat{S}_v$. This results in a recursive calculation of the total vertex demands $\hat{S}$ which can be computed by a depth-first search starting at the root vertex r . An outline of this algorithm is given in Algorithm 4.1.

Algorithm 4.1: BackwardSweep

Input: $G = (V, E, \phi_r, S, w)$ flow tree, $u \in V$

- 1 $\hat{S}_u \leftarrow S_u$
 - 2 **for** every child v of u , $e = (u, v)$ **do**
 - 3 BackwardSweep(G, v)
 - 4 $\hat{S}_u \leftarrow \hat{S}_u + bw(\phi_u, \hat{S}_v, w_e)$
-

4.2 Forward Sweep

In the *Forward Sweep*, we compute the vertex potential ϕ for all non-root vertices based on the estimated total vertex demands computed in the Backward Sweep. Because the forward function fw depends on the vertex potential of the parent, we start with the root vertex and move downwards to the leaves in a top-down manner. For every vertex v , we compute ϕ_v

by applying the forward function fw to the vertex potential of its parent u , the total vertex demand $\hat{S}_v$ computed in the Backward Sweep, and the edge parameter $w_{(u,v)}$. We update the vertex potential ϕ_v using the formula

$$\phi_v = fw(\phi_u, \hat{S}_v, w_{(u,v)})$$

for every vertex $v \in V \setminus \{r\}$. As in the Backward Sweep, this results in a recursive calculation of the vertex potentials ϕ which can be computed by a depth-first search starting at the root vertex r . An outline of this algorithm is given in Algorithm 4.2.

Algorithm 4.2: ForwardSweep

Input: $G = (V, E, \phi_r, S, w)$ flow tree, $u \in V$

```

1 for every child  $v$  of  $u$ ,  $e = (u, v)$  do
2    $\phi_v \leftarrow fw(\phi_u, \hat{S}_v, w_e)$ 
3   ForwardSweep( $G, v$ )
```

4.3 Backward-Forward Sweep

The *Backward-Forward Sweep (BFS)* [SHSL88] repeatedly executes the Backward Sweep and Forward Sweep to iteratively update the total vertex demands $\hat{S}$ and vertex potentials ϕ and improve their estimates. In every iteration, we first estimate the total vertex demands $\hat{S}$ in a Backward Sweep and then update the vertex potentials ϕ in a Forward Sweep. For this, we use the vertex potentials ϕ from the previous iteration. We repeat this process until the vertex potentials ϕ converge. Since the first iteration also depends on previous vertex potentials, we need to initialize the vertex potentials ϕ before the first iteration of the BFS. This results in the algorithm described by Algorithm 4.3.

Algorithm 4.3: BackwardForwardSweep

Input: $G = (V, E, \phi_r, S, w)$ flow tree rooted in r

```

1 for  $v \in V \setminus \{r\}$  do
2   initialize  $\phi_v$ 
3 while not converged do
4   BackwardSweep( $G, r$ )
5   ForwardSweep( $G, r$ )
```

We define one iteration of BFS as one execution of the Backward Sweep and Forward Sweep. This induces a sequence of vertex potentials $(\phi^{(i)} : V \rightarrow \mathbb{R}^n)_{i \in \mathbb{N}_0}$ and total vertex demands $(\hat{S}^{(i)} : V \rightarrow \mathbb{R}^n)_{i \in \mathbb{N}}$, where $\phi^{(i)}$ and $\hat{S}^{(i)}$ are the vertex potentials and total vertex demands after the i -th iteration. Furthermore, we define the initial vertex potentials $\phi^{(0)}$ as our initialized vertex values before the first iteration with $\phi_r^{(0)} = \phi_r$. How we choose the initial non-root vertex potentials $\phi^{(0)}$ depends on the flow functions f_p, f_c and affects the behavior of the BFS.

The structure of the Backward Sweep results in the formula

$$\hat{S}_u^{(i)} = S_u + \sum_{v \in c(u)} bw(\phi_u^{(i-1)}, \hat{S}_v^{(i)}, w_{(u,v)})$$

for updating the total vertex demands. The structure of the Forward Sweep results in the formula

$$\phi_v^{(i)} = fw(\phi_u^{(i)}, \hat{S}_v^{(i)}, w_{(u,v)})$$

for updating the vertex potentials if u is the parent of v .

We can see that the root potential ϕ_r is not updated in the Forward Sweep since the root has no parent, thus it does not change during a BFS iteration. As it is initialized with ϕ_r , we have $\phi_r^{(i)} = \phi_r^{(0)} = \phi_r$ for all $i \in \mathbb{N}$. Further, we can see that if u is a leaf vertex, u has no children, i.e., $c(u) = \emptyset$. Using this we can conclude that

$$\hat{S}_u^{(i)} = S_u + \sum_{v \in c(u)} bw(\phi_u^{(i-1)}, \hat{S}_v^{(i)}) = S_u .$$

This means that the total vertex demands $\hat{S}_u^{(i)}$ of leaf vertices u are equal to their initial vertex demands S_u and do not change during the BFS iterations.

Since the BFS is an algorithm to solve the Flow Problem, we want to show that the BFS does not change the vertex potentials ϕ if and only if they are a solution to the Flow Problem. This means, that we can stop the BFS if the vertex potentials ϕ do not change after an iteration, and conclude that they are a solution to the Flow Problem. For this, we first show that the total vertex demands $\hat{S}$ do not change during the next iteration if the vertex potentials ϕ are a solution to the Flow Problem.

Lemma 4.1 (Total Demand Equality): *If $\phi^{(i)}$ is a solution to the Flow Problem for some $i \in \mathbb{N}_0$, then the total vertex demands $\hat{S}^{(i)}$ do not change during the next iteration, that is $\hat{S}^{(i)} = \hat{S}^{(i+1)}$.*

Proof. For this proof, we define $\hat{S}^{(0)} = \hat{S}^{(1)}$ as the initial total vertex demands before the first iteration of the BFS. These can be chosen arbitrarily as they do not affect the result of any iteration. Let $(u, v) \in E$ be an edge and $i \in \mathbb{N}$. We use the definition of the Flow Problem and substitute $\phi_v^{(i)}$ for its definition.

$$\begin{aligned} S_v &= f_c(\phi_u^{(i)}, \phi_v^{(i)}) - \sum_{x \in c(v)} f_p(\phi_v^{(i)}, \phi_x^{(i)}) \\ \Leftrightarrow S_v &= f_c(\phi_u^{(i)}, fw(\phi_u^{(i)}, \hat{S}_v^{(i)}, w_{(u,v)})) - \sum_{x \in c(v)} f_p(\phi_v^{(i)}, \phi_x^{(i)}) \end{aligned}$$

Using the properties of fw .

$$\begin{aligned} \Leftrightarrow S_v &= \hat{S}_v^{(i)} - \sum_{x \in c(v)} f_p(\phi_v^{(i)}, \phi_x^{(i)}) \\ \Leftrightarrow S_v &= \hat{S}_v^{(i)} - \sum_{x \in c(v)} f_p(\phi_v^{(i)}, fw(\phi_v^{(i)}, \hat{S}_x^{(i)}, w_{(v,x)})) \end{aligned}$$

We use the definition of the backward function bw .

$$\begin{aligned} \Leftrightarrow S_v &= \hat{S}_v^{(i)} - \sum_{x \in c(v)} bw(\phi_v^{(i)}, \hat{S}_x^{(i)}) \\ \Leftrightarrow \hat{S}_v^{(i)} &= S_v + \sum_{x \in c(v)} bw(\phi_v^{(i)}, \hat{S}_x^{(i)}) \end{aligned}$$

We use this for a structural induction over a flow tree G to show that $\hat{S}^{(i)} = \hat{S}^{(i+1)}$. Let u be a leaf vertex. Then we have $\hat{S}_u^{(i)} = S_u = \hat{S}_u^{(i+1)}$. Now let u be a non-leaf vertex and assume that for all children v of u , we have $\hat{S}_v^{(i)} = \hat{S}_v^{(i+1)}$.

$$\begin{aligned}\hat{S}_u^{(i)} &= S_u + \sum_{v \in c(u)} bw(\phi_u^{(i)}, \hat{S}_v^{(i)}) \\ &= S_u + \sum_{v \in c(u)} bw(\phi_u^{(i)}, \hat{S}_v^{(i+1)}) \\ &= \hat{S}_u^{(i+1)}\end{aligned}$$

By structural induction, we have $\hat{S}^{(i)} = \hat{S}^{(i+1)}$. ■

We use this lemma to show that the BFS does not change the vertex potentials ϕ if and only if they are a solution to the Flow Problem.

Theorem 4.2 (BFS on Solutions): *If at any point during the BFS the vertex potentials do not change after an iteration, i.e., $\phi^{(i)} = \phi^{(i+1)}$ for some $i \in \mathbb{N}_0$, that is equivalent to $\phi^{(i)}$ being a solution to the Flow Problem, so the equation*

$$S_v = f_c(\phi_u^{(i)}, \phi_v^{(i)}) - \sum_{x \in c(v)} f_p(\phi_v^{(i)}, \phi_x^{(i)})$$

holds for every edge $(u, v) \in E$.

Proof. Assume that $\phi^{(i)} = \phi^{(i+1)}$ for some $i \in \mathbb{N}_0$. We show that the Flow Problem equation holds for every edge $(u, v) \in E$. We begin by using the definition of the Backward Sweep.

$$\begin{aligned}\hat{S}_v^{(i+1)} &= S_v + \sum_{x \in c(v)} bw(\phi_v^{(i)}, \hat{S}_x^{(i+1)}) \\ \Leftrightarrow S_v &= \hat{S}_v^{(i+1)} - \sum_{x \in c(v)} bw(\phi_v^{(i)}, \hat{S}_x^{(i+1)})\end{aligned}$$

We use our assumption $\phi^{(i)} = \phi^{(i+1)}$.

$$\Leftrightarrow S_v = \hat{S}_v^{(i+1)} - \sum_{x \in c(v)} bw(\phi_v^{(i+1)}, \hat{S}_x^{(i+1)})$$

Using the definition of the backward function bw , we can rewrite the second term on the right-hand side.

$$\Leftrightarrow S_v = \hat{S}_v^{(i+1)} - \sum_{x \in c(v)} f_p(\phi_v^{(i+1)}, fw(\phi_v^{(i+1)}, \hat{S}_x^{(i+1)}))$$

By definition of the Forward Sweep, we can substitute.

$$\Leftrightarrow S_v = \hat{S}_v^{(i+1)} - \sum_{x \in c(v)} f_p(\phi_v^{(i+1)}, \phi_x^{(i+1)})$$

We use the definition of the forward function.

$$\begin{aligned}\Leftrightarrow S_v &= f_c(\phi_u^{(i+1)}, fw(\phi_u^{(i+1)}, \hat{S}_v^{(i+1)})) - \sum_{x \in c(v)} f_p(\phi_v^{(i+1)}, \phi_x^{(i+1)}) \\ \Leftrightarrow S_v &= f_c(\phi_u^{(i+1)}, \phi_v^{(i+1)}) - \sum_{x \in c(v)} f_p(\phi_v^{(i+1)}, \phi_x^{(i+1)})\end{aligned}$$

Using our assumption once more.

$$\Leftrightarrow S_v = f_c(\phi_u^{(i)}, \phi_v^{(i)}) - \sum_{x \in c(v)} f_p(\phi_v^{(i)}, \phi_x^{(i)})$$

Since v was arbitrary, the Flow Problem is solved.

Now assume that $\phi^{(i)}$ is a solution to the Flow Problem for some $i \in \mathbb{N}_0$. We show that $\phi^{(i)} = \phi^{(i+1)}$ by structural induction over a flow tree G . For the root vertex r , we know that $\phi_r^{(i)} = \phi_r^{(i+1)}$. Now let v be a non-root vertex and assume that for its parent u , we have $\phi_u^{(i)} = \phi_u^{(i+1)}$.

$$\begin{aligned} \phi_v^{(i+1)} &= fw(\phi_u^{(i+1)}, \hat{S}_v^{(i+1)}) \\ &= fw(\phi_u^{(i)}, \hat{S}_v^{(i+1)}) \end{aligned}$$

We use Lemma 4.1 to conclude that $\hat{S}_v^{(i)} = \hat{S}_v^{(i+1)}$.

$$\begin{aligned} &= fw(\phi_u^{(i)}, \hat{S}_v^{(i)}) \\ &= \phi_v^{(i)} \end{aligned}$$

By structural induction, we have $\phi^{(i)} = \phi^{(i+1)}$. ■

Theorem 4.2 shows that solutions to the Flow Problem are the only fixed points of the BFS. This is a desirable property of the BFS since it guarantees that we do not skip any solutions. Furthermore, it allows us to stop the BFS as soon as the vertex potentials ϕ do not change after an iteration and return the found solution.

4.4 Convergence

In most cases, there is no i -th iteration of the BFS where the vertex potentials ϕ do not change after an iteration. Instead, the vertex potentials ϕ converge to a solution of the Flow Problem. To formalize this, we define the convergence of the BFS as follows and show that it is equivalent to the converged vertex potentials being a solution to the Flow Problem.

We say that the BFS converges if the limit of the vertex potentials $\lim_{i \rightarrow \infty} \phi_v^{(i)}$ is defined for every vertex v . We denote $\phi_v^{(\infty)} := \lim_{i \rightarrow \infty} \phi_v^{(i)}$ for every vertex v . Similarly, we define the limit of the total vertex demands $\lim_{i \rightarrow \infty} \hat{S}_v^{(i)}$ for every vertex v and denote $\hat{S}_v^{(\infty)} := \lim_{i \rightarrow \infty} \hat{S}_v^{(i)}$. To show that the converged vertex potentials $\phi^{(\infty)}$ are a solution to the Flow Problem, we first show that the total vertex demands $\hat{S}$ converge as well.

Observation 4.3 (Total Demand Convergence): *If the BFS converges, then the total vertex demands converge as well.*

Proof. By structural induction over a flow tree G . Let u be a leaf vertex. Then we have $\lim_{i \rightarrow \infty} \hat{S}_u^{(i)} = S_u$. Now let u be a non-leaf vertex and assume that for all children v of u , we have that $\lim_{i \rightarrow \infty} \hat{S}_v^{(i)} = \hat{S}_v^{(\infty)}$ is defined. Then we have

$$\begin{aligned} \lim_{i \rightarrow \infty} \hat{S}_u^{(i)} &= \lim_{i \rightarrow \infty} (S_u + \sum_{v \in c(u)} bw(\phi_u^{(i-1)}, \hat{S}_v^{(i)})) \\ &= S_u + \sum_{v \in c(u)} \lim_{i \rightarrow \infty} bw(\phi_u^{(i-1)}, \hat{S}_v^{(i)}) \\ &= S_u + \sum_{v \in c(u)} bw(\lim_{i \rightarrow \infty} \phi_u^{(i-1)}, \lim_{i \rightarrow \infty} \hat{S}_v^{(i)}) \\ &= S_u + \sum_{v \in c(u)} bw(\phi_u^{(\infty)}, \hat{S}_v^{(\infty)}) \end{aligned}$$

which is defined since $\phi_u^{(\infty)}$ and $\hat{S}_v^{(\infty)}$ are defined by the induction hypothesis. $\blacksquare$

We use this observation to show that the converged vertex potentials $\phi^{(\infty)}$ are a solution to the Flow Problem.

Theorem 4.4 (BFS Convergence Solution): *If the BFS converges, then the limit vertex potentials $\phi^{(\infty)}$ are a solution to the Flow Problem and so the equation*

$$S_v = f_c(\phi_u^{(\infty)}, \phi_v^{(\infty)}) - \sum_{x \in c(v)} f_p(\phi_v^{(\infty)}, \phi_x^{(\infty)})$$

holds for every $(u, v) \in E$.

Proof. If the BFS converges, then the limit of the vertex potentials $\phi_v^{(\infty)}$ and total vertex demands $\hat{S}_v^{(\infty)}$ are defined for every vertex v by Observation 4.3. By definition of the forward function, we can derive the following identity.

$$\phi_v^{(\infty)} = fw(\phi_u^{(\infty)}, \hat{S}_v^{(\infty)})$$

We use the definition of the Backward Sweep and rearrange.

$$\begin{aligned} \hat{S}_v^{(i+1)} &= S_v + \sum_{x \in c(v)} bw(\phi_v^{(i)}, \hat{S}_x^{(i+1)}) \\ \Leftrightarrow S_v &= \hat{S}_v^{(i+1)} - \sum_{x \in c(v)} bw(\phi_v^{(i)}, \hat{S}_x^{(i+1)}) \end{aligned}$$

Using the definition of the backward function bw , we can rewrite the second term on the right-hand side.

$$\Leftrightarrow S_v = \hat{S}_v^{(i+1)} - \sum_{x \in c(v)} f_p(\phi_v^{(i)}, fw(\phi_v^{(i)}, \hat{S}_x^{(i+1)}))$$

We use the definition of the forward function.

$$\Leftrightarrow S_v = f_c(\phi_u^{(i)}, fw(\phi_u^{(i)}, \hat{S}_v^{(i+1)})) - \sum_{x \in c(v)} f_p(\phi_v^{(i)}, fw(\phi_v^{(i)}, \hat{S}_x^{(i+1)}))$$

Since $\phi^{(i)}$ converges to $\phi^{(\infty)}$ and $\hat{S}^{(i)}$ converges to $\hat{S}^{(\infty)}$, we can take the limit on both sides.

$$\begin{aligned} \Leftrightarrow \quad S_v &= f_c(\phi_u^{(\infty)}, fw(\phi_u^{(\infty)}, \hat{S}_v^{(\infty)})) - \sum_{x \in c(v)} f_p(\phi_v^{(\infty)}, fw(\phi_v^{(\infty)}, \hat{S}_x^{(\infty)})) \\ \Leftrightarrow \quad S_v &= f_c(\phi_u^{(\infty)}, \phi_v^{(\infty)}) - \sum_{x \in c(v)} f_p(\phi_v^{(\infty)}, \phi_x^{(\infty)}) \end{aligned}$$

We can conclude that $\phi^{(\infty)}$ is a solution to the Flow Problem. ■

Theorem 4.4 shows that if the BFS converges, then the limit vertex potentials $\phi^{(\infty)}$ are a solution to the Flow Problem. But it does not show that the BFS converges. This strongly depends on the flow functions f_p, f_c and the initial vertex potentials $\phi^{(0)}$ as we will see in following chapters.

4.4.1 Epsilon Convergence

In practice, we do not compute the limit of the vertex potentials $\phi^{(\infty)}$, but instead stop the BFS if the vertex potentials ϕ do not change significantly after an iteration. That means, we choose a desired precision $\varepsilon > 0$ and stop the BFS if $|\phi_v^{(i)} - \phi_v^{(i-1)}| \leq \varepsilon$ for some $i \in \mathbb{N}$ and all $v \in V$. If such an i exists, we say that the BFS ε -converges to the vertex potentials $\phi^{(i)}$.

We take a look at the runtime of the BFS. The runtime of the BFS on a flow tree $G = (V, E, \phi_r, S, w)$ until ε -convergence is denoted by $T_{BFS}(G, \varepsilon)$. We can see that the Backward Sweep and Forward Sweep each visit every vertex $v \in V$ exactly once, which results in a runtime of $\mathcal{O}(|V|)$ for each sweep. Running the BFS for i iterations until ε -convergence results in a total runtime of $T_{BFS}(G, \varepsilon) \in \mathcal{O}(i \cdot |V|)$. We do not provide a bound on the number of iterations i until ε -convergence, and leave this as an open problem for future research.

5 Potential-Independent Backward Functions

While the BFS is executed the same for all flow functions f_p, f_c , its behavior strongly varies depending on the given functions. In this chapter we take a look at a class of functions, where the BFS solves the Flow Problem in one iteration. This class is defined by the property that the backward function bw is independent of the vertex potentials ϕ_u . We say a backward function bw is *potential-independent* if $bw(\phi_u, \hat{S}_v, w_e)$ is independent of the vertex potential ϕ_u , that is

$$bw(\phi_u, \hat{S}_v, w_e) = bw(\phi'_u, \hat{S}_v, w_e)$$

holds for every $\phi_u, \phi'_u, \hat{S}_v, w_e \in \mathbb{R}^n$. In this case, we can define an equivalent function $bw'(\hat{S}_v, w_e) := bw(\phi_u, \hat{S}_v, w_e)$. Since we only defined this property in terms of the backward function bw , it can be useful to show properties of the flow functions f_p, f_c that imply the potential-independence of bw .

Observation 5.1 (Potential-Independent Flow Functions): *If the outgoing flow function f_p is a function of the incoming flow function f_c and the edge parameter w_e , that is $f_p(\phi_u, \phi_v, w_e) = g(f_c(\phi_u, \phi_v, w_e), w_e)$ for some function $g : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$, then the backward function bw is potential-independent.*

Proof. We use the definition of the backward function bw .

$$\begin{aligned} bw(\phi_u, \hat{S}_v, w_e) &= f_p(\phi_u, fw(\phi_u, \hat{S}_v, w_e), w_e) \\ &= g(f_c(\phi_u, fw(\phi_u, \hat{S}_v, w_e), w_e), w_e) \\ &= g(\hat{S}_v, w_e) \end{aligned}$$

Therefore, the backward function bw is potential-independent, and we can define an equivalent function $bw'(\hat{S}_v, w_e) := g(\hat{S}_v, w_e)$. ■

Now that we have defined potential-independent backward functions and identified the conditions under which they arise, we can show that the BFS runs in linear time for such functions by showing that just one iteration of the BFS is sufficient to find a solution to the Flow Problem.

Theorem 5.2 (Potential-Independent Convergence): *If the backward functions bw is potential-independent, the vertex potentials after one iteration of the BFS $\phi^{(1)}$ are a solution to the Flow Problem.*

Proof. By structural induction over a flow tree G we can show that $\hat{S}^{(1)} = \hat{S}^{(2)}$. Let u be a leaf vertex. We know that $\hat{S}_u^{(1)} = S_u = \hat{S}_u^{(2)}$. Now let u be a non-leaf vertex and assume that for all children v of u , we have $\hat{S}_v^{(1)} = \hat{S}_v^{(2)}$. We conclude the following.

$$\hat{S}_u^{(1)} = S_u + \sum_{v \in c(u)} bw'(\hat{S}_v^{(1)}) = S_u + \sum_{v \in c(u)} bw'(\hat{S}_v^{(2)}) = \hat{S}_u^{(2)}$$

Therefore, we have $\hat{S}^{(1)} = \hat{S}^{(2)}$.

A final structural induction over a flow tree G shows that $\phi^{(1)} = \phi^{(2)}$. Let r be the root vertex. We know that $\phi_r^{(1)} = \phi_r^{(2)}$. Now let v be a non-root vertex and assume that for its parent u , we have $\phi_u^{(1)} = \phi_u^{(2)}$. With

$$\phi_v^{(1)} = fw(\phi_u^{(1)}, \hat{S}_v^{(1)}) = fw(\phi_u^{(2)}, \hat{S}_v^{(2)}) = \phi_v^{(2)}$$

we conclude that $\phi^{(1)} = \phi^{(2)}$.

Using Theorem 4.2 we can conclude $\phi^{(1)}$ is a solution to the Flow Problem. $\blacksquare$

With this result, it follows that the BFS runs in linear time for potential-independent backward functions, since it only needs one iteration to find a solution to the Flow Problem. That is, $T_{BFS}(G, \varepsilon) \in \mathcal{O}(i \cdot |V|) = \mathcal{O}(|V|)$.

5.1 Lossless Flow Functions

When choosing flow functions f_p, f_c an intuitive choice is to choose them such that no flow is lost when sent from one vertex to another. We will see that these flow functions form a subclass of potential-independent backward functions, and therefore the BFS finds a solution to the Flow Problem after one iteration. To formalize, a set of flow functions f_p, f_c is *lossless* if the flow sent from u to v is equal to the flow received at v from u for every edge $e = (u, v)$ and all edge parameters w , i.e., the equation

$$f_p(\phi_u, \phi_v, w_e) = f_c(\phi_u, \phi_v, w_e)$$

holds for all vertex potentials ϕ_u, ϕ_v and all edge parameters w_e . If no flow is lost when sent from one vertex to another, then the backward function bw should just return the total child vertex demand $\hat{S}_v$. We see this in the following observation.

Observation 5.3 (Lossless Backward Function): *If the flow functions f_p, f_c are lossless, then the backward function bw is equal to the total child vertex demand $\hat{S}_v$, that is*

$$bw(\phi_u, \hat{S}_v, w_e) = \hat{S}_v.$$

Proof. Using the definition of the backward function bw we have

$$bw(\phi_u, \hat{S}_v, w_e) = f_p(\phi_u, fw(\phi_u, \hat{S}_v, w_e), w_e) = f_c(\phi_u, fw(\phi_u, \hat{S}_v, w_e), w_e) = \hat{S}_v.$$

$\blacksquare$

From this, the next conclusion follows directly.

Corollary 5.4 (Lossless Convergence): *If the flow functions f_p, f_c are lossless, the vertex potentials after one iteration of the BFS $\phi^{(1)}$ are a solution to the Flow Problem.*

Proof. By Observation 5.3 we know that bw is potential-independent. Using Theorem 5.2 we can conclude $\phi^{(1)}$ is a solution to the Flow Problem. $\blacksquare$

As a result, we can conclude that the BFS runs in linear time for lossless flow functions, since it only needs one iteration to find a solution to the Flow Problem.

6 Counter-Monotonic Backward/Forward Functions

When looking at different flow functions f_p, f_c , we can find that some function classes have properties that allow us to solve the Flow Problem more easily. In Chapter 5 we looked at potential-independent backward functions which allow us to find a solution to the Flow Problem after one iteration of the BFS. In this chapter we look at another class of flow functions f_p, f_c that are not as easy to solve as potential-independent backward functions, but still have properties that allow us to show convergence of the BFS. That class of flow functions f_p, f_c is defined by a certain monotonicity property of the backward and forward functions bw, fw .

In detail, we require that the backward function bw is monotonically non-increasing in its first argument and monotonically non-decreasing in its second argument. That is, $bw(\phi_u, \hat{S}_v, w_e) \leq bw(\phi'_u, \hat{S}'_v, w_e)$ if $\phi_u \geq \phi'_u$ and $\hat{S}_v \leq \hat{S}'_v$. We also require that the forward function fw is monotonically non-decreasing in its first argument and monotonically non-increasing in its second argument. That is, $fw(\phi_u, \hat{S}_v, w_e) \leq fw(\phi'_u, \hat{S}'_v, w_e)$ if $\phi_u \leq \phi'_u$ and $\hat{S}_v \geq \hat{S}'_v$. We call such a set of backward and forward functions bw, fw *counter-monotonic* backward and forward functions. Further, the Flow Problem with counter-monotonic backward and forward functions is called the *Counter-Monotonic Flow Problem*. We restrict the Counter-Monotonic Flow Problem to be 1-dimensional, although the definitions can be extended to higher dimensions. Note that we do not state any requirements to the behavior of functions when the edge parameter w_e changes. Therefore, we omit the edge parameter as a function argument in the following definitions, observations and theorems.

First, we show that the vertex values, such as total demands and potentials, are also monotonic in the initial vertex potentials and vertex demands when the backward and forward functions bw, fw are counter-monotonic.

Lemma 6.1 (Monotonic Vertex Values): *Let G be an instance of the Counter-Monotonic Flow Problem. Let $S \leq S'$ be two assignments of vertex demands. If we initialize the vertex potential such that $\phi^{(0)} \geq \phi'^{(0)}$, and the BFS converges to a solution of the Flow Problem for both assignments of vertex demands S and S' , then the limit total vertex demands are less for S than for S' , that is $\hat{S}^{(\infty)} \leq \hat{S}'^{(\infty)}$. Also, the limit vertex potentials are greater for S than for S' , that is $\phi^{(\infty)} \geq \phi'^{(\infty)}$.*

Proof. We show by induction over i that $\hat{S}^{(i)} \leq \hat{S}'^{(i)}$ and $\phi^{(i)} \geq \phi'^{(i)}$. For the base case $i = 0$, we have $\phi^{(0)} \geq \phi'^{(0)}$ by assumption. For $i \in \mathbb{N}_0$, we show that $\hat{S}^{(i+1)} \leq \hat{S}'^{(i+1)}$ if $\phi^{(i)} \geq \phi'^{(i)}$ by structural induction over G . Let v be a leaf vertex.

$$\hat{S}_v^{(i+1)} = S_v \leq S'_v = \hat{S}'_v^{(i+1)}$$

For a non-leaf vertex v , we assume that for all children x of v , we have $\hat{S}_x^{(i+1)} \leq \hat{S}'_x^{(i+1)}$.

$$\hat{S}_v^{(i+1)} = S_v + \sum_{x \in c(v)} bw(\phi_v^{(i)}, \hat{S}_x^{(i+1)})$$

We use our assumption that $S \leq S'$.

$$\leq S'_v + \sum_{x \in c(v)} bw(\phi_v^{(i)}, \hat{S}_x^{(i+1)})$$

We use the monotonicity of bw twice.

$$\begin{aligned} &\leq S'_v + \sum_{x \in c(v)} bw(\phi_v'^{(i)}, \hat{S}_x^{(i+1)}) \\ &\leq S'_v + \sum_{x \in c(v)} bw(\phi_v^{(i)}, \hat{S}_x'^{(i+1)}) = \hat{S}_v'^{(i+1)} \end{aligned}$$

Now we show that $\phi_v^{(i+1)} \geq \phi_v'^{(i+1)}$ for every vertex v . Let r be the root vertex. We have $\phi_r^{(i+1)} \geq \phi_r'^{(i+1)}$ by assumption. Now let v be a non-root vertex and assume that for its parent u , we have $\phi_u^{(i+1)} \geq \phi_u'^{(i+1)}$. We use the monotonicity of the forward function fw and that $\hat{S}^{(i+1)} \leq \hat{S}'^{(i+1)}$.

$$\phi_v^{(i+1)} = fw(\phi_u^{(i)}, \hat{S}_v^{(i+1)}) \geq fw(\phi_u'^{(i)}, \hat{S}_v'^{(i+1)}) = \phi_v'^{(i+1)}$$

By induction, we have $\hat{S}^{(i)} \leq \hat{S}'^{(i)}$ and $\phi^{(i)} \geq \phi'^{(i)}$ for every $i \in \mathbb{N}$. Because of the continuity of the functions, the argument extends to the limit values, such that $\hat{S}^{(\infty)} \leq \hat{S}'^{(\infty)}$ and $\phi^{(\infty)} \geq \phi'^{(\infty)}$. $\blacksquare$

Lemma 6.1 shows an interesting connection between the vertex demands and the vertex potentials of a solution to the Flow Problem when the backward and forward functions bw, fw are counter-monotonic. But we did not yet show any convergence properties of the BFS when the backward and forward functions bw, fw are counter-monotonic. In the following theorem, we show an important property of the vertex potentials ϕ when running the BFS that will later make it easier for us to show convergence.

Theorem 6.2 (Monotonic Vertex Potentials): *Let G be an instance of the Counter-Monotonic Flow Problem. If the initial vertex potentials $\phi^{(0)}$ undersupply the total vertex demands, i.e.,*

$$f_c(\phi_u^{(0)}, \phi_v^{(0)}) \leq S_v + \sum_{x \in c(v)} f_p(\phi_v^{(0)}, \phi_x^{(0)})$$

for every non-root vertex v with parent u , then the vertex potentials decrease after every iteration, that is $\phi^{(i)} \geq \phi^{(i+1)}$ for every $i \in \mathbb{N}_0$.

Proof. We show that $\phi^{(i)} \geq \phi^{(i+1)}$ by induction over i . For the base case $i = 0$ we define $\hat{S}_v^{(0)} := f_c(\phi_u^{(0)}, \phi_v^{(0)})$ for every non-root vertex v with parent u . We show that $\hat{S}^{(0)} \leq \hat{S}^{(1)}$ by structural induction over G . For a leaf vertex v , we have $\hat{S}_v^{(0)} = \hat{S}_v^{(1)}$. Now let v be a non-leaf vertex and assume that for all children x of v , we have $\hat{S}_x^{(0)} \leq \hat{S}_x^{(1)}$. We use the monotonicity of the backward function bw and the definition of $\hat{S}_v^{(0)}$.

$$\begin{aligned} \hat{S}_v^{(1)} &= S_v + \sum_{x \in c(v)} bw(\phi_v^{(0)}, \hat{S}_x^{(1)}) \\ &\geq S_v + \sum_{x \in c(v)} bw(\phi_v^{(0)}, \hat{S}_x^{(0)}) \end{aligned}$$

Using the definition of the backward and forward functions.

$$\begin{aligned}
&= S_v + \sum_{x \in c(v)} f_p(\phi_v^{(0)}, fw(\phi_v^{(0)}, \hat{S}_x^{(0)})) \\
&= S_v + \sum_{x \in c(v)} f_p(\phi_v^{(0)}, fw(\phi_v^{(0)}, f_c(\phi_v^{(0)}, \phi_x^{(0)}))) \\
&= S_v + \sum_{x \in c(v)} f_p(\phi_v^{(0)}, \phi_x^{(0)})
\end{aligned}$$

We apply the theorem assumption.

$$\geq f_c(\phi_v^{(0)}, \phi_x^{(0)}) = \hat{S}_v^{(0)}$$

Therefore, we have $\hat{S}^{(0)} \leq \hat{S}^{(1)}$.

We show that $\phi^{(0)} \geq \phi^{(1)}$ by structural induction over G . Let r be the root vertex. We know that $\phi_r^{(0)} = \phi_r^{(1)}$. Now let v be a non-root vertex and assume that for its parent u , we have $\phi_u^{(0)} \geq \phi_u^{(1)}$. We use the definition of the forward function fw and the definition of $\hat{S}_v^{(0)}$, then apply the monotonicity of the forward function fw and that $\hat{S}^{(0)} \leq \hat{S}^{(1)}$.

$$\phi_v^{(0)} = fw(\phi_u^{(0)}, f_c(\phi_u^{(0)}, \phi_v^{(0)})) = fw(\phi_u^{(0)}, \hat{S}_v^{(0)}) \geq fw(\phi_u^{(1)}, \hat{S}_v^{(1)}) = \phi_v^{(1)}$$

Let $\phi^{(i-1)} \geq \phi^{(i)}$ for some $i \in \mathbb{N}$. We show that $\phi^{(i)} \geq \phi^{(i+1)}$. By structural induction over a flow tree G we can show that $\hat{S}^{(i)} \leq \hat{S}^{(i+1)}$. Let v be a leaf vertex. We have $\hat{S}_v^{(i)} = S_v = \hat{S}_v^{(i+1)}$. Now let v be a non-leaf vertex and assume that for all children x of v , we have $\hat{S}_x^{(i)} \leq \hat{S}_x^{(i+1)}$. We use the monotonicity of the backward function bw and the induction hypothesis $\phi^{(i-1)} \geq \phi^{(i)}$.

$$\hat{S}_v^{(i)} = S_v + \sum_{x \in c(v)} bw(\phi_v^{(i-1)}, \hat{S}_x^{(i)}) \leq S_v + \sum_{x \in c(v)} bw(\phi_v^{(i)}, \hat{S}_x^{(i+1)}) = \hat{S}_v^{(i+1)}$$

Therefore, we have $\hat{S}^{(i)} \leq \hat{S}^{(i+1)}$.

Another structural induction over a flow tree G shows that $\phi^{(i)} \geq \phi^{(i+1)}$. Let r be the root vertex. We have $\phi_r^{(i)} = \phi_r^{(i+1)}$. Now let v be a non-root vertex and assume that for its parent u , we have $\phi_u^{(i)} \geq \phi_u^{(i+1)}$. We use the monotonicity of the forward function fw and that $\hat{S}^{(i)} \leq \hat{S}^{(i+1)}$.

$$\phi_v^{(i)} = fw(\phi_u^{(i)}, \hat{S}_v^{(i)}) \geq fw(\phi_u^{(i+1)}, \hat{S}_v^{(i+1)}) = \phi_v^{(i+1)}$$

Therefore, we have $\phi^{(i)} \geq \phi^{(i+1)}$. ■

The result of Theorem 6.2 is very useful to show that the vertex potentials ϕ converge to a solution of the Flow Problem, since the only thing remaining is to show that the vertex potentials are bounded below. Then the vertex potentials ϕ converge to a solution of the Flow Problem by the Monotone Convergence Theorem.

Until now, we have only stated requirements for the backward and forward functions bw, fw to show that the vertex potentials ϕ decrease after every iteration of the BFS. Since the backward and forward functions bw, fw are defined in terms of the flow functions f_p, f_c , it is useful to find conditions on the flow functions f_p, f_c that guarantee the required monotonicity of the backward and forward functions bw, fw .

Observation 6.3 (Counter-Monotonic Flow Functions): *Let the incoming flow function f_c be monotonically non-decreasing in the first vertex potential ϕ_u and monotonically non-increasing in the second vertex potential ϕ_v , that is $f_c(\phi_u, \phi_v) \leq f_c(\phi'_u, \phi'_v)$ if $\phi_u \leq \phi'_u$ and $\phi_v \geq \phi'_v$. Further, let the outgoing flow function f_p be monotonically non-increasing in the both vertex potentials ϕ_u, ϕ_v , i.e., $f_p(\phi_u, \phi_v) \leq f_p(\phi'_u, \phi'_v)$ if $\phi_u \geq \phi'_u$ and $\phi_v \geq \phi'_v$. Then the backward and forward functions bw, fw are counter-monotonic.*

Proof. Assume that $\phi_u \leq \phi'_u$. We apply the monotonicity of f_c and the definition of the forward function fw .

$$f_c(\phi'_u, fw(\phi'_u, \hat{S}_v)) = \hat{S}_v = f_c(\phi_u, fw(\phi_u, \hat{S}_v)) \leq f_c(\phi'_u, fw(\phi_u, \hat{S}_v))$$

From this we can conclude that $fw(\phi_u, \hat{S}_v) \leq fw(\phi'_u, \hat{S}_v)$ because f_c is monotonically non-decreasing in its second argument. Assume that $\hat{S}_v \leq \hat{S}'_v$. We apply the monotonicity of f_c and the definition of the forward function fw .

$$f_c(\phi_u, fw(\phi_u, \hat{S}_v)) = \hat{S}_v \leq \hat{S}'_v = f_c(\phi_u, fw(\phi_u, \hat{S}'_v))$$

From this we can conclude that $fw(\phi_u, \hat{S}_v) \geq fw(\phi_u, \hat{S}'_v)$ because f_c is monotonically non-increasing in its second argument. We can conclude that the forward function fw is monotonically non-decreasing in its first argument ϕ_u and monotonically non-increasing in its second argument $\hat{S}_v$.

For the backward function bw , assume that $\phi_u \leq \phi'_u$. We apply the monotonicity of f_p and the definition of the backward function bw .

$$bw(\phi_u, \hat{S}_v) = f_p(\phi_u, fw(\phi_u, \hat{S}_v)) \geq f_p(\phi'_u, fw(\phi'_u, \hat{S}_v)) = bw(\phi'_u, \hat{S}_v)$$

Now assume that $\hat{S}_v \leq \hat{S}'_v$. We apply the monotonicity of f_p and the definition of the backward function bw .

$$bw(\phi_u, \hat{S}_v) = f_p(\phi_u, fw(\phi_u, \hat{S}_v)) \leq f_p(\phi_u, fw(\phi_u, \hat{S}'_v)) = bw(\phi_u, \hat{S}'_v)$$

From this we can conclude that the backward function bw is monotonically non-increasing in its first argument ϕ_u and monotonically non-decreasing in its second argument $\hat{S}_v$. Therefore, the backward and forward functions bw, fw are counter-monotonic. $\blacksquare$

These properties guarantee that the backward and forward functions are counter-monotonic. But we will see that other flow functions f_p, f_c can also have counter-monotonic backward and forward functions bw, fw .

6.1 Modified Power Flow

In literature, the BFS is often used to solve the *AC-Optimal Power Flow Problem* on radial networks. The *AC-OPF Problem* is an electrical engineering problem that aims to satisfy the power demands of a network [Cap16]. The vertex potentials $\phi_v \in \mathbb{C}$ represent complex voltages at a vertex v and the edge weights $w_e \in \mathbb{C}$ represent the complex admittance of an

edge e . The vertex demands $S_v \in \mathbb{C}$ represent the complex power demand at a vertex v . Then, the AC-OPF Problem on radial networks is equivalent to the 2-dimensional Flow Problem on flow trees with a set of flow functions we call *Power Flow* functions. For an edge $(u, v) \in E$, the Power Flow functions are defined as follows.

$$\begin{aligned} f_p(\phi_u, \phi_v, w_{(u,v)}) &= w_{(u,v)}^* \cdot (|\phi_u| - |\phi_v| \cdot e^{i\cdot\Theta}) \cdot |\phi_u| \\ f_c(\phi_u, \phi_v, w_{(u,v)}) &= w_{(u,v)}^* \cdot (|\phi_u| \cdot e^{-i\cdot\Theta} - |\phi_v|) \cdot |\phi_v| \end{aligned}$$

where $\Theta = \arg(\phi_u) - \arg(\phi_v)$ is the angle between the complex numbers ϕ_u and ϕ_v and $w_{(u,v)}^*$ is the complex conjugate of $w_{(u,v)}$.

We simplify the Power Flow functions by assuming that the vertex potentials ϕ_v are non-negative real numbers and the edge weights w_e are equal to 1. Let the vertex demands S_v also be non-negative real numbers. This eliminates the complex numbers from the Power Flow functions since the angle difference Θ is always zero and the complex conjugate of 1 is also 1. We define the flow functions f_p and f_c of the *Modified Power Flow* as follows.

$$\begin{aligned} f_p(\phi_u, \phi_v, w_{(u,v)}) &= (\phi_u - \phi_v) \cdot \phi_u \\ f_c(\phi_u, \phi_v, w_{(u,v)}) &= (\phi_u - \phi_v) \cdot \phi_v \end{aligned}$$

We can see that the flow functions are independent of the edge weights w_e , and thus we omit them in the following. The resulting forward and backward functions fw and bw of the Modified Power Flow are defined as follows.

$$\begin{aligned} fw(\phi_u, \hat{S}_v) &= \frac{\phi_u}{2} + \sqrt{\frac{\phi_u^2}{4} - \hat{S}_v} \\ bw(\phi_u, \hat{S}_v) &= \left(\frac{\phi_u}{2} - \sqrt{\frac{\phi_u^2}{4} - \hat{S}_v} \right) \cdot \phi_u \end{aligned}$$

We can verify that these functions are indeed the forward and backward functions of the flow functions f_p and f_c by showing that $f_c(\phi_u, fw(\phi_u, \hat{S}_v)) = \hat{S}_v$, $fw(\phi_u, f_c(\phi_u, \phi_v)) = \phi_v$, and $bw(\phi_u, \hat{S}_v) = f_p(\phi_u, fw(\phi_u, \hat{S}_v))$.

From now on, we assume that $S \geq 0$ such that every vertex is a consumer. Further, we demand that every feasible instance of the Flow Problem with Modified Power Flow functions has a solution where the vertex potentials are non-negative, i.e., $\phi \geq 0$.

The first observation we can make is that the total vertex demands $\hat{S}^{(i)}$ are non-negative for every iteration i of the BFS because the vertex demands S are non-negative.

Observation 6.4 (Non-Negative Total Vertex Demands): *Let the flow functions f_p and f_c be the flow functions of the Modified Power Flow and fw and bw be the corresponding forward and backward functions. If $S \geq 0$ and $\phi^{(i)} \geq 0$, then $\hat{S}^{(i)} \geq 0$ for every iteration i of the BFS.*

Proof. Let v be a leaf vertex. We know that $\hat{S}_v^{(i)} = S_v \geq 0$. Now let u be a non-leaf vertex and assume that for all children v of u , we have $\hat{S}_v^{(i+1)} \geq 0$.

$$\begin{aligned} \hat{S}_u^{(i+1)} &= S_u + \sum_{v \in c(u)} bw(\phi_u^{(i)}, \hat{S}_v^{(i+1)}) \\ &= S_u + \sum_{v \in c(u)} \left(\frac{\phi_u^{(i)}}{2} - \sqrt{\frac{\phi_u^{(i)2}}{4} - \hat{S}_v^{(i+1)}} \right) \cdot \phi_u^{(i)} \end{aligned}$$

Since $S_u \geq 0$ and $\hat{S}_v^{(i+1)} \geq 0$, the following inequality holds.

$$\geq \sum_{v \in c(u)} \left(\frac{\phi_u^{(i)}}{2} - \sqrt{\frac{\phi_u^{(i)2}}{4}} \right) \cdot \phi_u^{(i)} = 0$$

Since i was arbitrary, we have $\hat{S}^{(i)} \geq 0$ for every iteration i of the BFS. The argument also applies to the limit vertex potentials $\hat{S}^{(\infty)} \geq 0$. ■

From this observation we can derive some properties of the vertex potentials ϕ . First, we can show that the vertex potentials decrease when descending the tree from the root to the leaves. This is similar to flow in electrical networks, where current flows from higher to lower potentials.

Observation 6.5 (Decreasing Vertex Potentials): *Let the flow functions f_p and f_c be the flow functions of the Modified Power Flow and fw and bw be the corresponding forward and backward functions. For an edge $e = (u, v) \in E$ the parent vertex potential is greater than or equal to the child vertex potential in every iteration, that is $\phi_u^{(i)} \geq \phi_v^{(i)}$, if the vertex potentials are non-negative.*

Proof. We use the definition of the forward function fw , then we use Observation 6.4.

$$\phi_v^{(i)} = fw(\phi_u^{(i)}, \hat{S}_v^{(i)}) = \frac{\phi_u^{(i)}}{2} + \sqrt{\frac{\phi_u^{(i)2}}{4} - \hat{S}_v^{(i)}} \leq \frac{\phi_u^{(i)}}{2} + \sqrt{\frac{\phi_u^{(i)2}}{4}} = \phi_u^{(i)}$$

The same argument applies to the limit vertex potentials $\phi_u^{(\infty)} \geq \phi_v^{(\infty)}$. ■

Further, we observe that the forward and backward functions $fw(\phi, \hat{S}), bw(\phi, \hat{S})$ are only well-defined if $\frac{\phi^2}{4} \geq \hat{S}$. We conclude that if the vertex potentials ϕ are a solution to the Flow Problem with Modified Power Flow functions, then $\frac{\phi_u^2}{4} \geq \hat{S}_v$ for every edge $e = (u, v) \in E$.

In Chapter 6, we have shown that the convergence of the BFS is easily proven if the backward and forward functions are counter-monotonic. The following lemma shows that the backward and forward functions of the Modified Power Flow are indeed counter-monotonic.

Lemma 6.6 (Modified Power Flow Counter-Monotonicity): *The backward and forward functions of the Modified Power Flow are counter-monotonic.*

Proof. We show that $bw(\phi, \hat{S})$ is monotonically non-increasing in ϕ and monotonically non-decreasing in $\hat{S}$.

$$bw(\phi, \hat{S}) = \left(\frac{\phi}{2} - \sqrt{\frac{\phi^2}{4} - \hat{S}} \right) \cdot \phi = \frac{\phi^2}{2} - \phi \cdot \sqrt{\frac{\phi^2}{4} - \hat{S}}$$

Deriving with respect to ϕ gives us the following.

$$\frac{d}{d\phi} bw(\phi, \hat{S}) = \phi - \sqrt{\frac{\phi^2}{4} - \hat{S}} - \frac{\phi^2}{2 \cdot \sqrt{\frac{\phi^2}{4} - \hat{S}}}$$

Since $\hat{S} \geq 0$, the following inequality holds.

$$\begin{aligned} &\leq \phi - \sqrt{\frac{\phi^2}{4} - \frac{\phi^2}{2 \cdot \sqrt{\phi^2}}} \\ &= 0 \end{aligned}$$

Therefore, $bw(\phi, \hat{S})$ is monotonically non-increasing in ϕ . Let $\hat{S} \leq \hat{S}'$.

$$\begin{aligned} bw(\phi, \hat{S}) &= \frac{\phi^2}{2} - \phi \cdot \sqrt{\frac{\phi^2}{4} - \hat{S}} \\ &\leq \frac{\phi^2}{2} - \phi \cdot \sqrt{\frac{\phi^2}{4} - \hat{S}'} = bw(\phi, \hat{S}') \end{aligned}$$

Therefore, $bw(\phi, \hat{S})$ is monotonically non-decreasing in $\hat{S}$. Now we show that $fw(\phi, \hat{S})$ is monotonically non-decreasing in ϕ and non-increasing in $\hat{S}$. Let $\phi \leq \phi'$.

$$\begin{aligned} fw(\phi, \hat{S}) &= \frac{\phi}{2} + \sqrt{\frac{\phi^2}{4} - \hat{S}} \\ &\leq \frac{\phi'}{2} + \sqrt{\frac{\phi'^2}{4} - \hat{S}} = fw(\phi', \hat{S}) \end{aligned}$$

Therefore, $fw(\phi, \hat{S})$ is monotonically non-decreasing in ϕ . Let $\hat{S} \leq \hat{S}'$.

$$\begin{aligned} fw(\phi, \hat{S}) &= \frac{\phi}{2} + \sqrt{\frac{\phi^2}{4} - \hat{S}} \\ &\geq \frac{\phi}{2} + \sqrt{\frac{\phi^2}{4} - \hat{S}'} = fw(\phi, \hat{S}') \end{aligned}$$

Therefore, $fw(\phi, \hat{S})$ is non-increasing in $\hat{S}$. All together, we have shown that the backward and forward functions of the Modified Power Flow are counter-monotonic. $\blacksquare$

Since the backward and forward functions of the Modified Power Flow are counter-monotonic and thus fulfill the assumptions of Theorem 6.2, we can conclude that the vertex potentials decrease after every iteration of the BFS if the initial vertex potentials undersupply the total vertex demands. It remains to find such initial vertex potentials $\phi^{(0)}$ and show that the vertex potentials $(\phi^{(i)})_{i \in \mathbb{N}_0}$ are bounded below. Further, we have to show that the backward and forward functions are well-defined for every iteration i of the BFS.

Theorem 6.7 (Modified Power Flow Convergence): *Let f_p and f_c be the flow functions of the Modified Power Flow as defined above and fw and bw be the corresponding forward and backward functions. If we initialize the vertex potentials such that $\phi_v^{(0)} = \phi_r$ for every vertex v and a non-negative solution to the Flow Problem exists, then the BFS converges.*

Proof. By Lemma 6.6, the backward and forward functions of the Modified Power Flow are counter-monotonic and thus fulfill the assumptions of Theorem 6.2. We show that the initial vertex potentials undersupply the total vertex demands. Let v be a non-root vertex and u be its parent.

$$f_c(\phi_u^{(0)}, \phi_v^{(0)}) = f_c(\phi_r, \phi_r) = f_p(\phi_r, \phi_r) = (\phi_r - \phi_r) \cdot \phi_r = 0$$

Since there is no flow on any edge and the vertex demands are non-negative, all vertex demands are undersupplied.

$$f_c(\phi_u^{(0)}, \phi_v^{(0)}) = 0 \leq S_v = S_v + \sum_{x \in c(v)} f_p(\phi_v^{(0)}, \phi_x^{(0)})$$

Because the initial vertex potentials undersupply the total vertex demands we can apply Theorem 6.2 to conclude that the vertex potentials decrease after every iteration. It remains to show that the vertex potentials are bounded below. For this, we show that $\phi_v^{(i)} \geq \phi_v^*$ for every vertex v , where $\phi^* \geq 0$ are the vertex potential of a non-negative solution to the Flow Problem with Modified Power Flow functions. We show this by induction over the iterations i of the BFS.

If $i = 0$, we have $\phi_v^{(0)} = \phi_r = \phi_r^*$. From Observation 6.5, we know that the vertex potentials of a non-negative solution decrease when descending the tree from the root to the leaves. Therefore, $\phi_v^{(0)} \geq \phi_v^*$ holds for every vertex v .

Now assume that $i \geq 0$ and that $\phi_v^{(i)} \geq \phi_v^*$ holds for every vertex v . First, we show that $\hat{S}^{(i+1)} \leq \hat{S}^*$, where $\hat{S}^*$ are the total vertex demands of the non-negative solution ϕ^* . Let v be a leaf vertex. We have $\hat{S}_v^{(i+1)} = S_v = \hat{S}_v^*$. Now let u be a non-leaf vertex and assume that for every child v of u , we have $\hat{S}_v^{(i+1)} \leq \hat{S}_v^*$. We use the counter-monotonicity of the backward function bw and the induction hypothesis.

$$\hat{S}_u^{(i+1)} = S_u + \sum_{v \in c(u)} bw(\phi_u^{(i)}, \hat{S}_v^{(i+1)}) \leq S_u + \sum_{v \in c(u)} bw(\phi_u^*, \hat{S}_v^*) = \hat{S}_u^*$$

For the root vertex r , we have $\phi_r^{(i+1)} = \phi_r = \phi_r^*$. Let v be a non-root vertex and assume that for its parent u , we have $\phi_u^{(i+1)} \geq \phi_u^*$. We use the definition of the forward function fw and $\hat{S}^{(i+1)} \leq \hat{S}^*$. Because fw is counter-monotonic, we can conclude the following.

$$\phi_v^{(i+1)} = fw(\phi_u^{(i+1)}, \hat{S}_v^{(i+1)}) \geq fw(\phi_u^*, \hat{S}_v^*) = \phi_v^*$$

Finally, we must show that the backward and forward functions are well-defined for every iteration i of the BFS. We have $\phi_u^{(i)} \geq \phi_u^*$ and $\hat{S}_v^{(i)} \leq \hat{S}_v^*$ for every edge $e = (u, v)$. For a non-negative solution ϕ^* , we have $\frac{\phi_u^{*2}}{4} \geq \hat{S}_v^*$. From this follows that $\frac{\phi_u^{(i)2}}{4} \geq \hat{S}_v^{(i)}$ and $\frac{\phi_u^{(i)2}}{4} \geq \hat{S}_v^{(i+1)}$. Therefore, the backward and forward functions are well-defined for every iteration i of the BFS.

Finally, we know that for every vertex v the potentials $\phi_v^{(0)} \geq \phi_v^{(1)} \geq \dots \geq \phi_v^*$ converge to a limit $\phi_v^{(\infty)} \geq \phi_v^*$ and therefore to a solution to the Flow Problem as proven in Theorem 4.4. ■

From this we can conclude that the BFS converges for the Modified Power Flow functions if we initialize the vertex potentials such that $\phi_v^{(0)} = \phi_r$ for every vertex v and a non-negative solution exists. Therefore, running the BFS on feasible instances of the Flow Problem with Modified Power Flow functions, that is instances where a solution with non-negative vertex potentials exists, will always converge to a solution. We sum this up in the following corollary.

Corollary 6.8 (Modified Power Flow Convergence): *A non-negative solution to the Flow Problem with Modified Power Flow functions exists if and only if the BFS converges when initialized with $\phi_v^{(0)} = \phi_r$ for every vertex v . Furthermore, if the BFS converges, it converges to the largest non-negative solution.*

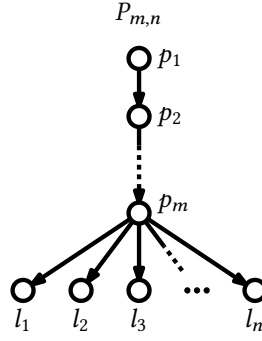


Figure 6.1: A palm tree $P_{m,n}$ of height m and width n with root vertex p_1 . The vertices $p_1, \dots, p_m$ form a directed path and the vertices $l_1, \dots, l_n$ are leaves attached to the end of the path.

Proof. Assume that a non-negative solution to the Flow Problem with Modified Power Flow functions exists. Then, by Theorem 6.7, the BFS converges when initialized with $\phi_v^{(0)} = \phi_r$ for every vertex v . Further, that solution is the largest non-negative, since we chose the solution ϕ^* in Theorem 6.7 arbitrarily and $\phi^{(\infty)} \geq \phi^*$.

Now assume that no non-negative solution to the Flow Problem with Modified Power Flow functions exists. We can show that the vertex potentials $\phi^{(i)}$ of the BFS are still bounded below by 0.

$$\phi_v^{(i)} = fw(\phi_u^{(i)}, \hat{S}_v^{(i)}) = \frac{\phi_u^{(i)}}{2} + \sqrt{\frac{\phi_u^{(i)2}}{4} - \hat{S}_v^{(i)}} \geq \frac{\phi_u^{(i)}}{2} \geq 0$$

This means that the vertex potentials $\phi^{(i)}$ would converge to a limit $\phi^{(\infty)} \geq 0$ and therefore a solution if the forward and backward functions are well-defined in every iteration. We conclude, that there must be an iteration i where the backward function is not well-defined for an edge, that is $\frac{\phi_u^{(i)2}}{4} < \hat{S}_v^{(i+1)}$ for some edge $e = (u, v)$. ■

This corollary shows that we can use the BFS with Modified Power Flow functions to solve any instance of the Flow Problem with Modified Power Flow. If no solution exists, we will detect this by the fact that the backward function is not well-defined for some edge in some iteration of the BFS and can conclude that no solution exists.

6.2 Palm Trees

In this section, we take a closer look at a subset of flow trees that are particularly easy to analyze. We define a *palm tree* $P_{m,n} = (V, E)$ of height $m \geq 2$ and width $n \geq 1$ as a directed tree with the following properties. The tree is composed of a directed path of length m and n leaves attached to the end of the path. This results in $V = \{p_1, \dots, p_m\} \cup \{l_1, \dots, l_n\}$ and $E = \{(p_i, p_{i+1}) \mid 1 \leq i < m\} \cup \{(p_m, l_j) \mid 1 \leq j \leq n\}$. An illustration of a palm tree $P_{m,n}$ is shown in Figure 6.1.

We now focus on flow trees $G = (V, E, \phi_r, S, w)$ where the underlying directed tree is a palm tree $P_{m,n}$. Furthermore, we assume all non-leaf vertex demands to be zero, i.e., $S_{p_i} = 0$ for every $1 \leq i \leq m$. The leaves all have the same demand $S_{l_j} = s/n$ for every $1 \leq j \leq n$ and some $s > 0$. Since all leaves have the same demand, we can conclude that their vertex potentials are equal, i.e., $\phi_{l_j} = \phi_l$ for every $1 \leq j \leq n$.

We are trying to find the maximum value for s such that the instance is still feasible. Because the demand at every path vertex p_i is zero, the incoming flow at p_i is equal to the total outgoing flow to p_{i+1} for $1 \leq i < m$. From this, we can conclude that the difference between the vertex potentials of two consecutive path vertices is constant.

$$\begin{aligned} f_c(\phi_{p_{i-1}}, \phi_{p_i}) &= f_p(\phi_{p_i}, \phi_{p_{i+1}}) \\ \Leftrightarrow (\phi_{p_{i-1}} - \phi_{p_i}) \cdot \phi_{p_i} &= (\phi_{p_i} - \phi_{p_{i+1}}) \cdot \phi_{p_i} \end{aligned}$$

We divide both sides by ϕ_{p_i} , which we assume to not be 0, since this is only possible if $s = 0$. If $s = 0$ the solution is trivial with $\phi_v = \phi_r$ for every vertex v .

$$\Leftrightarrow \phi_{p_{i-1}} - \phi_{p_i} = \phi_{p_i} - \phi_{p_{i+1}}$$

We denote the difference between the vertex potentials of two consecutive path vertices by $\delta := \phi_{p_i} - \phi_{p_{i+1}}$ for $1 \leq i < m$. From this, we can derive the vertex potentials of the path vertices in terms of ϕ_r and δ . The potential at each path vertex p_i is $\phi_{p_i} = \phi_r - (i - 1) \cdot \delta$ for $1 \leq i \leq m$. Further, we can derive the vertex potentials of the leaf vertices in terms of ϕ_r and δ .

Observation 6.9 (Leaf Vertex Potentials): *The solution vertex potential at each leaf vertex l_j is $\phi_l = \phi_r - (m + \frac{1}{n} - 1) \cdot \delta$.*

Proof. For p_m the incoming and outgoing flow must be equal, since the demand at p_m is zero.

$$\begin{aligned} f_c(\phi_{p_{m-1}}, \phi_{p_m}) &= \sum_{j=1}^n f_p(\phi_{p_m}, \phi_{l_j}) \\ \Leftrightarrow (\phi_{p_{m-1}} - \phi_{p_m}) \cdot \phi_{p_m} &= \sum_{j=1}^n (\phi_{p_m} - \phi_{l_j}) \cdot \phi_{p_m} \end{aligned}$$

We divide both sides by ϕ_{p_m} , which is not 0 for the same reason as above.

$$\Leftrightarrow \phi_{p_{m-1}} - \phi_{p_m} = \sum_{j=1}^n (\phi_{p_m} - \phi_{l_j})$$

Since all leaf vertex potentials are equal, we can simplify the right-hand side and substitute the left-hand side for δ .

$$\begin{aligned} \Leftrightarrow \delta &= n \cdot (\phi_{p_m} - \phi_l) \\ \Leftrightarrow \phi_l &= \phi_{p_m} - \frac{\delta}{n} \end{aligned}$$

We substitute ϕ_{p_m} with $\phi_r - (m - 1) \cdot \delta$.

$$\Leftrightarrow \phi_l = \phi_r - \left(m + \frac{1}{n} - 1\right) \cdot \delta$$

■

Now that we know all vertex potentials in terms of ϕ_r and δ , we can derive the maximal value for s such that a solution to the Flow Problem exists.

Theorem 6.10 (Maximum Value of s): *The maximal value for s such that a solution to the Flow Problem exists is $s_{\max} = \frac{\phi_r^2}{4 \cdot (m + \frac{1}{n} - 1)}$.*

Proof. The incoming flow at each leaf l_j is calculated as follows.

$$f_c(\phi_{p_m}, \phi_l) = (\phi_{p_m} - \phi_l) \cdot \phi_l = \frac{\delta}{n} \cdot \left(\phi_r - \left(m + \frac{1}{n} - 1 \right) \cdot \delta \right)$$

To maximize the incoming flow and thus s , we can derive the incoming flow with respect to δ and set it equal to zero.

$$0 \stackrel{!}{=} \frac{d}{d\delta} \left(\frac{\delta}{n} \cdot \left(\phi_r - \left(m + \frac{1}{n} - 1 \right) \cdot \delta \right) \right)$$

We apply the product rule to the left-hand side.

$$\Leftrightarrow 0 = \frac{1}{n} \cdot \left(\phi_r - \left(m + \frac{1}{n} - 1 \right) \cdot \delta \right) + \frac{\delta}{n} \cdot \left(- \left(m + \frac{1}{n} - 1 \right) \right)$$

Multiplying both sides by n and rearranging gives us the following.

$$\Leftrightarrow 0 = \phi_r - 2 \cdot \left(m + \frac{1}{n} - 1 \right) \cdot \delta$$

We solve for δ .

$$\Leftrightarrow \delta = \frac{\phi_r}{2 \cdot \left(m + \frac{1}{n} - 1 \right)}$$

Substituting this value for δ into the expression for the incoming flow shows us that this is indeed a maximum. We can now calculate the maximal value for s .

$$\begin{aligned} s_{\max} &= n \cdot f_c(\phi_{p_m}, \phi_l) \\ &= n \cdot \frac{\delta}{n} \cdot \left(\phi_r - \left(m + \frac{1}{n} - 1 \right) \cdot \delta \right) \\ &= \delta \cdot \left(\phi_r - \left(m + \frac{1}{n} - 1 \right) \cdot \delta \right) \end{aligned}$$

We substitute the value for δ into the expression.

$$\begin{aligned} &= \frac{\phi_r}{2 \cdot \left(m + \frac{1}{n} - 1 \right)} \cdot \left(\phi_r - \left(m + \frac{1}{n} - 1 \right) \cdot \frac{\phi_r}{2 \cdot \left(m + \frac{1}{n} - 1 \right)} \right) \\ &= \frac{\phi_r}{2 \cdot \left(m + \frac{1}{n} - 1 \right)} \cdot \left(\phi_r - \frac{\phi_r}{2} \right) \\ &= \frac{\phi_r^2}{4 \cdot \left(m + \frac{1}{n} - 1 \right)} \end{aligned}$$

This concludes the proof that the maximal value for s such that a solution to the Flow Problem exists is $s_{\max} = \frac{\phi_r^2}{4 \cdot (m + \frac{1}{n} - 1)}$. Therefore, every instance with $s > s_{\max}$ must be infeasible and no solution exists. ■

The upper bound for s derived in Theorem 6.10 reveals that the maximum demand $s_{\max}$ is proportional to the square of the root potential ϕ_r . Further, $s_{\max}$ decreases with increasing height m of the palm tree. This is intuitive, as a longer path from the root to the leaves has more edges and thus more losses, which reduces the maximum demand that can be satisfied at the leaves. On the other hand, $s_{\max}$ increases with increasing width n of the palm tree. This is due to the fact that the total demand is distributed among more leaves, which reduces the demand at each individual leaf and thus allows for a higher total demand to be satisfied.

7 Non-Tree Graphs

When only looking at directed trees, the Flow Problem is not as complex as on general rooted directed graphs because every non-root vertex has exactly one parent. That means that the total demand $\hat{S}$ of a vertex must be completely satisfied by the incoming flow from its only parent. When looking at all DAGs, the total vertex demand $\hat{S}$ can be supplied by multiple parents. The added complexity lies in the question of how to distribute the total demand $\hat{S}$ among the incoming flows from the parents.

We have to redefine some of the definitions from Chapter 3 to account for this added complexity. First, we define a *flow DAG* $G = (V, E, \phi_r, S, w)$ similarly to flow trees, but now (V, E) is a DAG instead of a directed tree. Further, the excess flow at a vertex v is now the sum of the incoming flows minus the sum of outgoing flows and thus calculated as

$$\sum_{u \in p(v)} S_{v \leftarrow u} - \sum_{x \in c(v)} S_{v \rightarrow x}.$$

Our goal is still to find vertex potentials such that the excess flow at each vertex satisfies its demand exactly. So, given a flow DAG $G = (V, E, \phi_r, S, w)$ and flow functions f_p, f_c , the *Flow Problem* is to find vertex potentials ϕ such that for every vertex $v \in V$ the equation

$$S_v = \sum_{u \in p(v)} S_{v \leftarrow u} - \sum_{x \in c(v)} S_{v \rightarrow x}$$

holds and the root potential equals ϕ_r . By using the definitions of the send and receive functions, we can rewrite this equation as

$$S_v = \sum_{u \in p(v)} f_c(\phi_u, \phi_v, w_{(u,v)}) - \sum_{x \in c(v)} f_p(\phi_v, \phi_x, w_{(v,x)})$$

to get an equivalent definition depending on the vertex potentials ϕ .

7.1 Routing

To solve the question of how to distribute the total demand $\hat{S}$ among the incoming flows from the parents, we introduce a *routing function* or *routing* ρ . This function determines the fraction of the total demand $\hat{S}$ of a child v that is supplied by an edge from a parent u . To formalize this, we define a routing function $\rho : E \rightarrow [0, 1]$ such that for every non-root vertex v the total vertex demand $\hat{S}$ is fully distributed among the incoming flows from the parents, i.e., $\sum_{u \in p(v)} \rho((u, v)) = 1$. We will shorten $\rho((u, v))$ to $\rho(u, v)$ for better readability.

From this definition, we can conclude that $\rho(e) = 1$ for all edges e in a flow tree, since every non-root vertex has exactly one parent and thus the total demand $\hat{S}$ is fully supplied by the incoming flow from its only parent. For flow DAGs, the routing function ρ can be used to distribute the total demand $\hat{S}$ among the incoming flows from the parents. A visualization of an example routing function is shown in Figure 7.1.

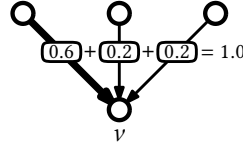


Figure 7.1: Example of a routing function ρ for a flow DAG. The vertex v has three parents from which it receives flow. Its first parent supplies most of the total demand $\hat{S}_v$ as suggested by the size of the incoming edge. The routing values add up to 1.

We now use the routing function ρ to generalize the definition of the forward and backward functions fw, bw from Chapter 3 to account for the distribution of the total demand $\hat{S}$. Since ρ determines the fraction of the total demand $\hat{S}$ of a child v that is supplied by an edge $e = (u, v)$, we know that the following equation must hold for a solution of the Flow Problem.

$$f_c(\phi_u, \phi_v, w_e) = \rho(e) \cdot \hat{S}_v$$

We can again solve this for the vertex potential ϕ_v of the child v and define the forward function fw for flow DAGs.

$$fw(\phi_u, \rho(e) \cdot \hat{S}_v, w_e) = \phi_v$$

As in Chapter 3, the forward function fw and the receive function f_c are inverses in the second argument and satisfy the following equations.

$$f_c(\phi_u, fw(\phi_u, \rho(e) \cdot \hat{S}_v, w_e), w_e) = \rho(e) \cdot \hat{S}_v, \quad fw(\phi_u, f_c(\phi_u, \phi_v, w_e), w_e) = \phi_v$$

Then, we can use the forward function fw to define the backward function bw for flow DAGs.

$$bw(\phi_v, \rho(e) \cdot \hat{S}_v, w_e) = f_p(\phi_v, fw(\phi_u, \rho(e) \cdot \hat{S}_v, w_e), w_e)$$

Note that these definitions are very similar to the definitions of the forward and backward functions fw, bw for flow trees and in the next sections, we will see that they also behave similarly. Still, we cannot yet solve the Flow Problem on flow DAGs since the BFS from Chapter 4 breaks down in the Forward Sweep when we calculate multiple potentials ϕ_v for a vertex v with multiple parents. Because the calculated potentials can be different, simply setting ϕ_v to one of them can lead to a state where the forward function fw does not apply to the other incoming edges. This problem extends to the limit vertex potentials as well, which then are no longer a solution to the Flow Problem. We solve this problem by introducing the concept of a split graph in the next section.

7.2 Split Graph

Flow DAGs are more complex than flow trees because a vertex can have multiple parents. We can simplify the problem by converting a flow DAG G into a flow tree G° by splitting every vertex with multiple parents. We call such a vertex a *join vertex*. To convert G into a flow tree, we define *splitting* G at a vertex v . We denote this operation as $G \circ v$ and define it as follows. Let G_v be the induced subgraph of v in G . For each parent $u_1, \dots, u_k$ of v , we create a copy $G_v^1 \dots G_v^k$ of the induced subgraph G_v . The split $G \circ v$ is then defined as the graph $(G \setminus G_v) \cup G_v^1 \cup \dots \cup G_v^k$ with edges (u_i, v_i) for all $1 \leq i \leq k$, that is, we remove the induced subgraph G_v and connect each parent u_i to its corresponding copy v_i of the original vertex v .

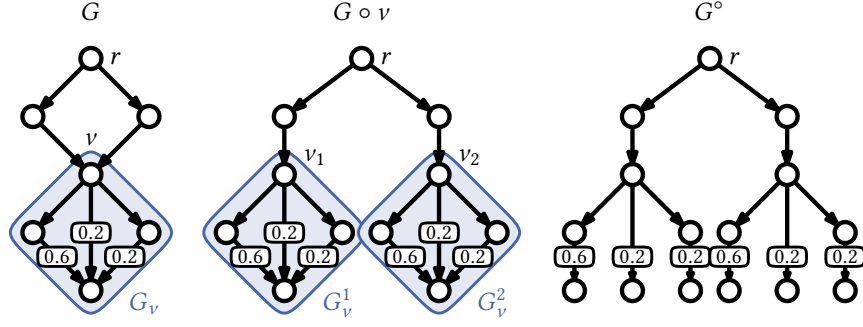


Figure 7.2: Example of a flow DAG G with a join vertex v and its induced subgraph G_v . The graph $G \circ v$ obtained by splitting the join vertex v . The split graph G° is obtained by splitting all remaining join vertices in $G \circ v$. The graphs show how the routing function ρ is extended to $G \circ v$ and G° .

We call the vertices $v_1, \dots, v_k$ the *split vertices* of v . The demands and edge parameters of the resulting graph are copied from the original graph G . If a routing ρ is given, we can define ρ° on $G \circ v$ by defining $\rho^\circ(u_i, v_i) := \rho(u_i, v)$ for all $1 \leq i \leq k$ and copying the routing for all other edges. Note that ρ° does not sum to one for the incoming edges of the split vertices v_i and is therefore not a routing. However, $\sum_{i=1}^k \rho^\circ(u_i, v_i) = 1$ still holds.

We define the *split graph* G° of a flow DAG G as the graph obtained by repeatedly splitting vertices with multiple parents until every non-root vertex has exactly one parent. We do this in bottom-up order to avoid cloning an unsplit join vertex. An example of a flow DAG G and its split graph G° is shown in Figure 7.2. The split graph is unique for a given flow DAG G and is a flow tree. Note that the size of the split graph can be exponential in the size of the original flow DAG G since every vertex with multiple parents and its induced subgraph are cloned for each parent.

We use the split graph to define a modified Backward-Forward Sweep that can solve the Flow Problem on flow DAGs when a correct routing is given.

7.3 Routed Backward-Forward Sweep

Given a flow DAG $G = (V, E, \phi_r, S, w)$, flow functions f_p, f_c , and a routing function ρ , we can use the routed forward and backward functions fw, bw from Section 7.1 to construct an iterative algorithm that solves the Flow Problem. We modify the Backward-Forward Sweep from Chapter 4 to incorporate the routing ρ . Furthermore, in the Forward Sweep, we must additionally account for the fact that a vertex can now have multiple parents since the regular Forward Sweep only calculates the vertex potential based on one parent. We fix this by preprocessing the flow DAG G into its split graph G° thereby ensuring that every vertex has at most one parent.

The Backward Sweep is still performed in the same way as before, but now we use the backward function bw for routed flows to calculate the total vertex demand $\hat{S}_u^{(i+1)}$ of a vertex u in iteration $i + 1$. This results in the formula

$$\hat{S}_u^{(i+1)} = S_u + \sum_{v \in c(u)} bw(\phi_u^{(i)}, \rho^\circ(u, v) \cdot \hat{S}_v^{(i+1)}, w_{(u,v)})$$

for every non-root vertex u .

Since we preprocessed the flow DAG into the split graph G° , the Forward Sweep is also performed in the same way as before, but using the forward function fw for routed flows to calculate the vertex potential $\phi_v^{(i+1)}$ of the child v in iteration $i + 1$.

$$\phi_v^{(i+1)} = fw(\phi_u^{(i+1)}, \rho^\circ(u, v) \cdot \hat{S}_v^{(i+1)}, w_{(u,v)})$$

We do this for every non-root vertex v with parent u .

The *Routed Backward-Forward Sweep (RBFS)* is defined as the splitting of the flow DAG G into the split graph G° followed by the repeated execution of the modified Backward and Forward Sweep. We execute these sweeps until the vertex potentials $\phi^{(i)}$ converge to a solution of the Flow Problem. As in the Backward-Forward Sweep, we must initialize the vertex potentials $\phi^{(0)}$ before the first iteration.

We define the limit of the vertex potentials $\phi^{(\infty)}$ and total vertex demands $\hat{S}^{(\infty)}$ as with the regular BFS, that is, if the limit values exist for every vertex of the split graph G° , we define $\phi^{(\infty)} := \lim_{i \rightarrow \infty} \phi^{(i)}$ and $\hat{S}^{(\infty)} := \lim_{i \rightarrow \infty} \hat{S}^{(i)}$. Also, we define the ε -convergence of the RBFS in the same way as with the regular BFS and get a runtime of $T_{RBFS}(G, \varepsilon) = T_{BFS}(G^\circ, \varepsilon) \in \mathcal{O}(i \cdot |V^\circ|)$ where $|V^\circ|$ is the number of vertices in the split graph G° and i is the number of iterations until ε -convergence.

Since the RBFS runs on the split graph G° and not on the original flow DAG G , we will later show how we can use the limit values $\phi^{(\infty)}$ and $\hat{S}^{(\infty)}$ to get a solution of the Flow Problem on the original flow DAG G . As with the regular BFS, we can show that if the RBFS converges, it converges to a solution of the Flow Problem. The proof is similar to the proof of Theorem 4.4 but uses the definitions of the backward and forward functions bw, fw for flow DAGs.

Theorem 7.1 (RBFS Convergence Solution): *If the RBFS with routing ρ converges and for every vertex v in G with split vertices $v_1, \dots, v_k$ in G° we have $\phi_{v_1}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$ and $\hat{S}_{v_1}^{(\infty)} = \dots = \hat{S}_{v_k}^{(\infty)}$, then the limit vertex potentials $\phi^{(\infty)}$ give us a solution to the Flow Problem on the original flow DAG G , that is, the equation*

$$S_v = \sum_{u \in p(v)} f_c(\phi_u^{(\infty)}, \phi_v^{(\infty)}) - \sum_{x \in c(v)} f_p(\phi_v^{(\infty)}, \phi_x^{(\infty)})$$

holds for every $v \in V$.

Proof. If the BFS converges, then the limit of the vertex potentials $\phi_v^{(\infty)}$ and total vertex demands $\hat{S}_v^{(\infty)}$ are defined for every vertex v . We can derive the following identity for an edge (u, v) .

$$\phi_v^{(\infty)} = fw(\phi_u^{(\infty)}, \rho^\circ(u, v) \cdot \hat{S}_v^{(\infty)})$$

Let v be a vertex in G with split vertices $v_1, \dots, v_k$ in the split graph G° . We use the fact that ρ is a routing and ρ° is defined as $\rho^\circ(u_i, v_i) := \rho(u_i, v)$. We define $\phi_v^{(\infty)} := \phi_{v_1}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$ and $\hat{S}_v^{(\infty)} := \hat{S}_{v_1}^{(\infty)} = \dots = \hat{S}_{v_k}^{(\infty)}$. This means that the potential of the vertex v in the original flow DAG G is defined as the potential of its split vertices v_i in the split graph G° .

$$\begin{aligned} \hat{S}_v^{(\infty)} &= \sum_{i=0}^k \rho(u_i, v) \cdot \hat{S}_v^{(\infty)} = \sum_{i=0}^k \rho^\circ(u_i, v_i) \cdot \hat{S}_{v_i}^{(\infty)} \\ &= \sum_{i=0}^k f_c(\phi_{u_i}^{(\infty)}, fw(\phi_{u_i}^{(\infty)}, \rho^\circ(u_i, v_i) \cdot \hat{S}_{v_i}^{(\infty)})) \\ &= \sum_{i=0}^k f_c(\phi_{u_i}^{(\infty)}, \phi_{v_i}^{(\infty)}) \end{aligned}$$

We translate this back to the original flow DAG G .

$$= \sum_{u \in p(v)} f_c(\phi_u^{(\infty)}, \phi_v^{(\infty)})$$

Now, we use the definition of the Backward Sweep and rearrange.

$$\begin{aligned} \hat{S}_v^{(i+1)} &= S_v + \sum_{x \in c(v)} bw(\phi_v^{(i)}, \rho^\circ(v, x) \cdot \hat{S}_x^{(i+1)}) \\ \Leftrightarrow S_v &= \hat{S}_v^{(i+1)} - \sum_{x \in c(v)} bw(\phi_v^{(i)}, \rho^\circ(v, x) \cdot \hat{S}_x^{(i+1)}) \end{aligned}$$

Using the definition of the backward function bw , we can rewrite the second term on the right-hand side.

$$\Leftrightarrow S_v = \hat{S}_v^{(i+1)} - \sum_{x \in c(v)} f_p(\phi_v^{(i)}, fw(\phi_v^{(i)}, \rho^\circ(v, x) \cdot \hat{S}_x^{(i+1)}))$$

Since $\phi^{(i)}$ converges to $\phi^{(\infty)}$ and $\hat{S}^{(i)}$ converges to $\hat{S}^{(\infty)}$, we can take the limit on both sides.

$$\begin{aligned} S_v &= \hat{S}_v^{(\infty)} - \sum_{x \in c(v)} f_p(\phi_v^{(\infty)}, fw(\phi_v^{(\infty)}, \rho^\circ(v, x) \cdot \hat{S}_x^{(\infty)})) \\ \Leftrightarrow S_v &= \sum_{u \in p(v)} f_c(\phi_u^{(\infty)}, \phi_v^{(\infty)}) - \sum_{x \in c(v)} f_p(\phi_v^{(\infty)}, \phi_x^{(\infty)}) \end{aligned}$$

We can conclude that a solution to the Flow Problem on the original flow DAG G can be obtained from the limit vertex potentials $\phi^{(\infty)}$ of the RBFS on the split graph G° by choosing an arbitrary copy of a vertex and using its potential in the original DAG G . $\blacksquare$

When including the routing function ρ and using the forward and backward functions for routed flows, the other previous results such as the solution property of Theorem 4.2, the properties of potential-independent and counter-monotonic functions of Chapter 5 and Chapter 6 can be similarly proven for the Routed Backward-Forward Sweep as well. Also, properties of the Modified Power Flow functions of Section 6.1 such as Theorem 6.7 and Corollary 6.8 still hold for the Routed Backward-Forward Sweep. The modified proofs are similar to the original proofs, but substitute the total vertex demands $\hat{S}$ with the routed total vertex demands $\rho \cdot \hat{S}$ and use that the routing function adds to one for the incoming edges of a vertex.

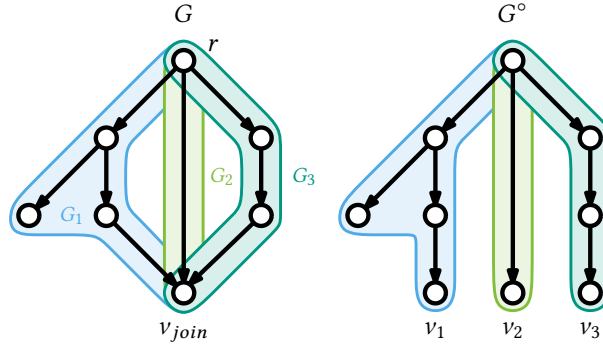


Figure 7.3: A disjoint flow DAG G with a join vertex v_{join} and the three components G_1, G_2, G_3 . Its split graph G° shows the split components.

In Theorem 7.1, we stated the condition that the limit vertex potentials and total demands must be equal for all split vertices of a join vertex v , that is, $\phi_{v_1}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$ and $\hat{S}_{v_1}^{(\infty)} = \dots = \hat{S}_{v_k}^{(\infty)}$. This is not satisfied for all possible routings. We call a routing ρ that satisfies this condition a *correct routing*. Since the routing function ρ is not part of an instance of the Flow Problem, we must first find a correct routing ρ for a given flow DAG $G = (V, E, \phi_r, S, w)$ and then use the Routed Backward-Forward Sweep to solve the Flow Problem on G . In the following, we show how to do this for a subclass of flow DAGs.

7.4 Disjoint Flow DAGs

In this section, we construct an algorithm to solve the Flow Problem on a restricted class of flow DAGs to later use this algorithm as a subroutine in a more general algorithm. Let $G = (V, E, \phi_r, S, w)$ be a flow DAG with flow functions f_p, f_c and counter-monotonic backward and forward functions bw, fw . We restrict G to have only one join vertex v_{join} with k parents $u_1, \dots, u_k$ and no children. Further, we demand that all paths from the root r to v_{join} are pairwise disjoint, that is, they share no edges. We call this a *disjoint flow DAG*. The i -th path from the root r to v_{join} is the path that contains u_i . A disjoint flow DAG where the join vertex v_{join} has k parents admits k components $G_1, \dots, G_k$ that are connected only by the root r and the join vertex v_{join} . If $(r, u_i, \dots, v_{join})$ is the i -th path from the root r to the join vertex v_{join} , then the component G_i is defined as the union of the root r , the edge (r, u_i) , and the induced subgraph G_{u_i} of the vertex u_i following the root on the i -th path. On the split graph G° , the component G_i contains the split vertex v_i instead of the join vertex v_{join} . An example of a disjoint flow DAG and its split graph is shown in Figure 7.3. Because these components share only the root r which has a fixed potential ϕ_r , and the join vertex v_{join} which has a fixed total demand $\hat{S}_{v_{join}}$, the behavior of the RBFS for a given routing on each component G_i is independent of the other components.

To solve the Flow Problem on a disjoint flow DAG $G = (V, E, \phi_r, S, w)$, we split G at the join vertex v_{join} to get its split graph G° . Note that the size of G° is only linear in the size of G since we only split one vertex with no children. Because the instance of the Flow Problem does not provide a routing, we observe some properties of ρ that help us find a correct routing. First, we show that the limit vertex potential of a split vertex v_i is monotonic in the value of $\rho(u_i, v_{join})$ and therefore independent of the other routing values. In detail, we show that

if we increase the routing value $\rho(u_i, v_{join})$ for a parent u_i of the join vertex v_{join} , then the vertex potential $\phi_{v_i}^{(\infty)}$ of the corresponding split vertex v_i in the split graph G° decreases. The proof for this is very similar to that of Lemma 6.1 because increasing $\rho(u_i, v_{join})$ is essentially the same as increasing the total vertex demand.

Lemma 7.2 (Routing Monotonicity): *Let G be an instance of the Counter-Monotonic Flow Problem and a disjoint flow DAG with join vertex v_{join} with parents $u_1, \dots, u_k$ and let G° be its split graph. Let ρ, ρ' be two routings on G and $\phi^{(i)}, \phi'^{(i)}$ be the vertex potentials during the RBFS with routing ρ and ρ' , respectively. If the RBFS converges and is initialized equally for both routings, then $\phi_{v_j}^{(i)} \leq \phi'_{v_j}{}^{(i)}$ if and only if $\rho(u_j, v_{join}) \geq \rho'(u_j, v_{join})$.*

Proof. Assume $\rho(u_j, v_{join}) \geq \rho'(u_j, v_{join})$. By the definition of the split graph, we have $\rho^\circ(u_j, v_j) \geq \rho'^\circ(u_j, v_j)$. Since the components $G_1, \dots, G_k$ of G are independent, we can analyze the behavior of the RBFS on the component G_j separately. Therefore, we only look at vertices v of G_j and show that $\phi_v^{(i)} \leq \phi'_v{}^{(i)}$. For $i = 0$, we have $\phi^{(0)} \leq \phi'^{(0)}$ by assumption that the RBFS is initialized equally for both routings. Assume that $\phi_v^{(i)} \leq \phi'_v{}^{(i)}$ holds for some iteration i . We first show that $\hat{S}_u^{(i+1)} \geq \hat{S}'_u{}^{(i+1)}$ holds for the total vertex demands in iteration $i + 1$. For this, we perform a structural induction over the vertices of G_j . Let u be a leaf. We have $\hat{S}_u^{(i+1)} = S_u = \hat{S}'_u{}^{(i+1)}$. Now, let u be a non-leaf vertex of G_j and assume that $\hat{S}_v^{(i+1)} \geq \hat{S}'_v{}^{(i+1)}$ holds for all children v of u .

$$\hat{S}_u^{(i+1)} = S_u + \sum_{v \in c(u)} bw(\phi_v^{(i)}, \rho^\circ(u, v) \cdot \hat{S}_v^{(i+1)})$$

We use the counter-monotonicity of the backward function bw and the induction hypothesis.

$$\geq S_u + \sum_{v \in c(u)} bw(\phi_v^{(i)}, \rho'^\circ(u, v) \cdot \hat{S}'_v{}^{(i+1)}) = \hat{S}'_u{}^{(i+1)}$$

Analogously, we can show that $\phi_u^{(i+1)} \leq \phi'_u{}^{(i+1)}$ holds for the vertex potentials in iteration $i + 1$.

$$\phi_u^{(i+1)} = fw(\phi_p^{(i+1)}, \rho^\circ(p, u) \cdot \hat{S}_u^{(i+1)})$$

We use the counter-monotonicity of the forward function fw and the induction hypothesis.

$$\leq fw(\phi_p^{(i+1)}, \rho'^\circ(p, u) \cdot \hat{S}'_u{}^{(i+1)}) = \phi'_u{}^{(i+1)}$$

Therefore, we have $\phi_v^{(i)} \leq \phi'_v{}^{(i)}$ for all vertices v of G_j and by taking the limit, we have $\phi_v^{(\infty)} \leq \phi'_v{}^{(\infty)}$. Swapping the roles of ρ and ρ' , we get that $\phi_{v_j}^{(\infty)} \geq \phi'_{v_j}{}^{(\infty)}$ if $\rho(u_j, v_{join}) \leq \rho'(u_j, v_{join})$. The equivalence follows directly from these two statements. $\blacksquare$

This lemma shows that the vertex potential $\phi_{v_j}^{(\infty)}$ can be controlled by increasing or decreasing $\rho(u_j, v_j)$. That means, we can guess a routing ρ and then use the RBFS to figure out how to modify the routing values to get a correct routing. We use this, to find a routing where all potentials of the split vertices $v_1, \dots, v_k$ are equal. Because these vertices are leaves, their total vertex demands are always equal and such a routing is already correct.

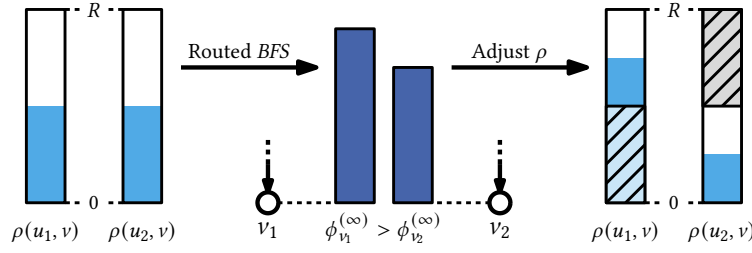


Figure 7.4: One iteration of the binary search for finding a balanced routing on a disjoint flow DAG with join vertex v . $\rho(u_1, v)$ is set to the midpoint of the current search interval and $\rho(u_2, v)$ is set to the remaining value. After running the RBFS with this routing, we compare the vertex potentials $\phi_{v_1}^{(\infty)}$ and $\phi_{v_2}^{(\infty)}$ to decide whether to increase or decrease $\rho(u_1, v)$.

7.4.1 Routing Search Algorithm

We base our algorithm on the following idea. Assume we have two split vertices v_1, v_2 and know the sum of the routing values $R := \rho(u_1, v_{\text{join}}) + \rho(u_2, v_{\text{join}})$ but not how to distribute this sum among the two edges (u_1, v_{join}) and (u_2, v_{join}) . By using the monotonicity of the vertex potentials $\phi_{v_1}^{(\infty)}$ and $\phi_{v_2}^{(\infty)}$ in the routing values we can find a routing where $\phi_{v_1}^{(\infty)} = \phi_{v_2}^{(\infty)}$. We do this by performing a binary search on the routing value $\rho(u_1, v_{\text{join}}) \in [0, R]$ and assigning the remaining value to $\rho(u_2, v_{\text{join}}) = R - \rho(u_1, v_{\text{join}})$. To decide whether to increase or decrease $\rho(u_1, v_{\text{join}})$, we can compare the vertex potentials $\phi_{v_1}^{(\infty)}$ and $\phi_{v_2}^{(\infty)}$ after having run a RBFS with the current routing. If $\phi_{v_1}^{(\infty)} < \phi_{v_2}^{(\infty)}$, Lemma 7.2 tells us that we must decrease $\rho(u_1, v_{\text{join}})$ to increase $\phi_{v_1}^{(\infty)}$. Simultaneously, this will increase $\rho(u_2, v_{\text{join}})$ and thus decrease $\phi_{v_2}^{(\infty)}$. An illustration of one iteration of this binary search is shown in Figure 7.4.

If there exists a distribution of the routing values such that $\phi_{v_1}^{(\infty)} = \phi_{v_2}^{(\infty)}$, then this binary search will find it. On the other hand, if no distribution of the routing values exists such that $\phi_{v_1}^{(\infty)} = \phi_{v_2}^{(\infty)}$, then the binary search will converge to a routing where $\rho(u_1, v_{\text{join}}) = 0$ or $\rho(u_2, v_{\text{join}}) = 0$. This happens when the routing sum R is too small. We will later prove these statements.

The search is an infinite process that converges to a correct routing. In practice, we can stop the search when the difference between the routing of the last iteration and the current iteration is smaller than a given threshold $\varepsilon_{\text{bin}} > 0$. Note that this is not directly related to the ε -convergence of the BFS and RBFS from Section 4.4.1, and we must choose ε_{bin} and ε_{bfs} independently. The relationship between ε_{bin} and ε_{bfs} is not trivial and is left open for future work.

If the join vertex v_{join} has exactly two parents, we can set $R := 1$ and get a correct routing. We generalize this to a join vertex v_{join} with k parents by recursively applying the binary search to match the vertex potentials of more and more split vertices. For a parent vertex u_i , the idea is to fix all previous routing values and finding a distribution of the remaining routing values such that $\phi_{v_i}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$. Then, we step to the previous parent vertex u_{i-1} and find a distribution of the remaining routing values such that $\phi_{v_{i-1}}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$. By repeating this process, we ultimately find a correct routing.

Beginning with the first split vertex v_1 , we know that the sum of the routing values is $R_1 := 1$. Therefore, we binary search for a routing value $\rho(u_1, v_{\text{join}}) \in [0, R_1]$ such that $\phi_{v_1}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$. For the i -th split vertex v_i , the sum of the remaining routing values must be

$R_i := R_{i-1} - \rho(u_{i-1}, v_{join})$, and we try to binary search for a routing value $\rho(u_i, v_{join}) \in [0, R_i]$ such that $\phi_{v_i}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$. We do this very similarly to the binary search for two split vertices. Assume that we already have $\phi_{v_{i+1}}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$, and we want to find a routing value $\rho(u_i, v_{join})$ to match the vertex potential $\phi_{v_i}^{(\infty)}$ to the others. If $\phi_{v_i}^{(\infty)} < \phi_{v_{i+1}}^{(\infty)}$, Lemma 7.2 tells us that we must decrease $\rho(u_i, v_{join})$ to increase $\phi_{v_i}^{(\infty)}$. This will simultaneously increase the sum of the remaining routing values $R_{i+1} = R_i - \rho(u_i, v_{join})$. Since the rest routing has changed, we must find a new distribution of the remaining routing values to match the vertex potentials $\phi_{v_{i+1}}^{(\infty)}, \dots, \phi_{v_k}^{(\infty)}$. If at any point we do not find a distribution of the remaining routing values such that $\phi_{v_{i+1}}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$, we must decrease $\rho(u_i, v_{join})$ because the remaining routing R_{i+1} is too small. Repeating this process will lead to a distribution of R_i such that $\phi_{v_i}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$. This results in Algorithm 7.1 which we call the *Routing Search* on a disjoint flow DAG G .

Algorithm 7.1: RoutingSearch

Input: $G = (V, E, \phi_r, S, w)$ disjoint flow DAG, routing ρ , $i \leq k$

```

1  if  $i = k$  then
2     $\rho(u_k, v_{join}) \leftarrow R_k$ 
3    RoutedBFS( $G, \rho$ )
4  else
5    low  $\leftarrow 0$ , high  $\leftarrow R_i$ 
6    while  $\phi_i^{(\infty)} \neq \phi_{i+1}^{(\infty)}$  do
7       $\rho(u_i, v_{join}) \leftarrow (low + high)/2$ 
8      RoutingSearch( $G, i + 1$ )
9      if  $\phi_i^{(\infty)} < \phi_{i+1}^{(\infty)}$  then
10       high  $\leftarrow \rho(u_i, v_{join})$ 
11     else
12       low  $\leftarrow \rho(u_i, v_{join})$ 

```

It remains to show the correctness of the Routing Search. We assume that all RBFs converge. For this we take a closer look at the remaining routing R_i . First, we show that the vertex potentials of a correct distribution of the remaining routing R_i are monotonic in R_i .

Lemma 7.3 (Remaining Routing Monotonicity): *Let G be an instance of the Counter-Monotonic Flow Problem and a disjoint flow DAG with join vertex v_{join} with parents $u_1, \dots, u_k$. Furthermore, let R_i, R'_i be two sums of routing values for a fixed i and let $\phi^{(\infty)}, \phi'^{(\infty)}$ be limit vertex potentials with routing values R_i and R'_i , respectively. If we find a distribution for both R_i and R'_i such that $\phi_{v_i}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$ and $\phi'_{v_i}{}^{(\infty)} = \dots = \phi'_{v_k}{}^{(\infty)}$, then the following equivalence holds. $\phi_{v_i}^{(\infty)} \geq \phi'_{v_i}{}^{(\infty)}$ if and only if $R_i \leq R'_i$.*

Proof. Assume $R_i \leq R'_i$. The definition of the remaining routing guarantees that $R_i = \sum_{j=i}^k \rho(u_j, v_{join})$ and $R'_i = \sum_{j=i}^k \rho'(u_j, v_{join})$. Because $R_i \leq R'_i$, there must exist a j such that $\rho(u_j, v_{join}) \leq \rho'(u_j, v_{join})$. By Lemma 7.2, we have $\phi_{v_j}^{(\infty)} \geq \phi'_{v_j}{}^{(\infty)}$. Since we have $\phi_{v_i}^{(\infty)} = \phi_{v_j}^{(\infty)}$ and $\phi'_{v_i}{}^{(\infty)} = \phi'_{v_j}{}^{(\infty)}$, we can conclude that $\phi_{v_i}^{(\infty)} \geq \phi'_{v_i}{}^{(\infty)}$. Swapping the roles of R_i and R'_i , we get the other direction of the equivalence. $\blacksquare$

Now, we explore the case that there is no distribution of the remaining routing R_i such that $\phi_{v_i}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$. In this case, we can want to show that the remaining routing R_i is too small.

Lemma 7.4 (Small Remaining Routing): *Let G be an instance of the Counter-Monotonic Flow Problem and a disjoint flow DAG with join vertex v_{join} with parents $u_1, \dots, u_k$. If there is no distribution of the remaining routing R_i such that $\phi_{v_i}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$, then the remaining routing R_i is too small.*

Proof. Assume that there is no distribution of the remaining routing R_i such that $\phi_{v_i}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$. That is, for every value of $\rho(u_i, v_{join}) \in [0, R_i]$, we have $\phi_{v_i}^{(\infty)} \neq \phi_{v_{i+1}}^{(\infty)}$. Because of the continuity of the values, two cases can occur. First, $\phi_{v_i}^{(\infty)} < \phi_{v_{i+1}}^{(\infty)}$ for all $\rho(u_i, v_{join}) \in [0, R_i]$. Therefore, this also holds for $\rho(u_i, v_{join}) = 0$ where $\phi_{v_i}^{(\infty)}$ is maximal and $\phi_{v_{i+1}}^{(\infty)}$ is minimal for all distributions of R_i by Lemma 7.2. Second, $\phi_{v_i}^{(\infty)} > \phi_{v_{i+1}}^{(\infty)}$ for all $\rho(u_i, v_{join}) \in [0, R_i]$. Again, this also holds for $\rho(u_i, v_{join}) = R_i$ where $\phi_{v_i}^{(\infty)}$ is minimal and $\phi_{v_{i+1}}^{(\infty)}$ is maximal for all distributions of R_i . In both cases, we can conclude that the remaining routing R_i must be increased to find a distribution such that $\phi_{v_i}^{(\infty)} = \phi_{v_{i+1}}^{(\infty)}$. ■

Note that if no correct distribution of the remaining routing R_i exists, the Routing Search will converge to a routing where $\rho(u_i, v_{join}) = 0$ or $\rho(u_i, v_{join}) = R_i$ since the comparison of the vertex potentials $\phi_{v_i}^{(\infty)}$ and $\phi_{v_{i+1}}^{(\infty)}$ will always yield the same result. With this, we show that if a correct distribution of the remaining routing R_i exists, and we can solve $\phi_{v_{i+1}}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$ recursively, then we find the correct distribution of the remaining routing such that $\phi_{v_i}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$.

Lemma 7.5 (Routing Subsearch): *Let G be an instance of the Counter-Monotonic Flow Problem and a disjoint flow DAG with join vertex v_{join} with parents $u_1, \dots, u_k$. Assume a distribution of R_i exists such that $\phi_{v_i}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$ and we can find a correct distribution of R_{i+1} or detect if none exists for all $R_{i+1} \in [0, R_i]$. Then, the Routing Search finds such a distribution of the remaining routing R_i .*

Proof. We know that a correct distribution of R_i exists such that $\phi_{v_i}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$. Then the routing value of the edge (u_i, v_{join}) must satisfy $\rho(u_i, v_{join}) \in [0, R_i]$. Let $\rho(u_i, v_{join})$ be the current routing value and $\rho^*(u_i, v_{join})$ be the correct value.

Assume, we can find a distribution of the remaining routing $R_{i+1} = R_i - \rho(u_i, v_{join})$ such that $\phi_{v_{i+1}}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$. If $\phi_{v_i}^{(\infty)} < \phi_{v_{i+1}}^{(\infty)}$, we can conclude that $\rho(u_i, v_{join}) > \rho^*(u_i, v_{join})$ by Lemma 7.2 and Lemma 7.3. Thereby, we eliminate all routing values greater than the current value. Analogously, if $\phi_{v_i}^{(\infty)} > \phi_{v_{i+1}}^{(\infty)}$, we can conclude that $\rho(u_i, v_{join}) < \rho^*(u_i, v_{join})$ and eliminate the lower values.

Now assume, there is no distribution of the remaining routing R_{i+1} such that $\phi_{v_{i+1}}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$. By Lemma 7.4, R_{i+1} is too small, so we have $\rho(u_i, v_{join}) > \rho^*(u_i, v_{join})$.

Choosing the midpoint of the remaining interval and repeating this process will converge to the correct routing value $\rho^*(u_i, v_{join})$. ■

This gives us all we need to show the correctness of the algorithm.

Theorem 7.6 (Routing Search Correctness): *The Routing Search finds a correct routing ρ for a disjoint flow DAG G with join vertex v_{join} with parents $u_1, \dots, u_k$ if and only if such a routing exists.*

Proof. For $i = k - 1$ we find a correct distribution of R_{k-1} if one exists by binary searching $\rho(u_{k-1}, v_{join}) \in [0, R_{k-1}]$. This follows from Lemma 7.2, Lemma 7.3 and the correctness of the decisions in the binary search. If no distribution exists, Lemma 7.4 shows how to detect this. We use Lemma 7.5 as the induction step and conclude that we can correctly distribute R_1 if such a distribution exists.

Therefore, we find a routing such that $\phi_{v_1}^{(\infty)} = \dots = \phi_{v_k}^{(\infty)}$. The limit total vertex demand of all split vertices is equal to S_v . Theorem 7.1 tells us that this is a correct routing ρ and the RBFS converges to a solution. If no correct routing exists, the Routing Search cannot find one and Lemma 7.4 tells us how we can detect this case. ■

With the correctness of the Routing Search, we can now solve the Flow Problem on a disjoint flow DAG G by finding a correct routing ρ . The resulting vertex potentials when the Routing Search finds a correct routing ρ are then a solution to the Flow Problem on G by Theorem 7.1.

At the beginning of this section, we assumed that every RBFS converges. When using the Modified Power Flow functions of Section 6.1, this does not pose a problem since Corollary 6.8 guarantees convergence if a solution exists. If not, we know that the backward or forward functions are not defined for an edge e and some iteration i . Let that edge e lie in the j -th component and with Lemma 7.2, we can conclude that the routing value $\rho(u_j, v_{join})$ is too big which we can use in the routing search.

We take a look at the runtime of the Routing Search. If we run the binary search for each routing value $\rho(u_i, v_{join})$ with a precision of ε_{bin} , the RBFS runs $\mathcal{O}(\log(\frac{1}{\varepsilon_{bin}}))$ times. Because we have k parents of the join vertex v_{join} , we nest the binary search $k - 1$ times and run the RBFS for each iteration. This results in a total runtime of $\mathcal{O}(\log^{k-1}(\frac{1}{\varepsilon_{bin}}) \cdot T_{RBFS}(G, \varepsilon))$ where ε is the precision of the RBFS.

7.5 Nested Disjoint Flow DAGs

In the previous section, we showed how to solve the Flow Problem on a disjoint flow DAG G with exactly one join vertex v_{join} . A question we answer by solving a disjoint flow DAG is what the total demand at the root r is if we know the total demand at the join vertex v_{join} and the root potential ϕ_r . This is exactly the question the backward function bw answers for a single edge as seen in Figure 4.1. So, we can abstract the disjoint flow DAG G with join vertex v_{join} to a single edge (r, v_{join}) and use the algorithm as a blackbox that solves the Flow Problem on this abstracted subgraph. Whenever we would normally call the backward function bw for the edge (r, v_{join}) , we instead solve the Flow Problem on the disjoint flow DAG G and get the correct total demand at the root r . An illustration of this is shown in Figure 7.5.

Calculating the vertex potentials in the next Forward Sweep will then update the root potential of G , and we have to solve the Flow Problem on G again. We will see that this abstraction can be used to solve the Flow Problem on a wider class of flow DAGs, we call *nested disjoint flow DAGs* or *nested disjoint*.

An inductive definition of a nested disjoint reveals why we are interested in this class of flow DAGs. Every flow tree is a nested disjoint. If G is a nested disjoint and (u, v) is an edge in G , then the flow DAG G' that is obtained by replacing the edge (u, v) with a disjoint flow DAG with join vertex v and root u is also a nested disjoint. An example of such a construction is shown in Figure 7.6. We denote the initial flow tree as G_0 , and note that G_0 is unique for every nested disjoint G . Any edge (u, v) that is replaced in this process admits a unique smaller

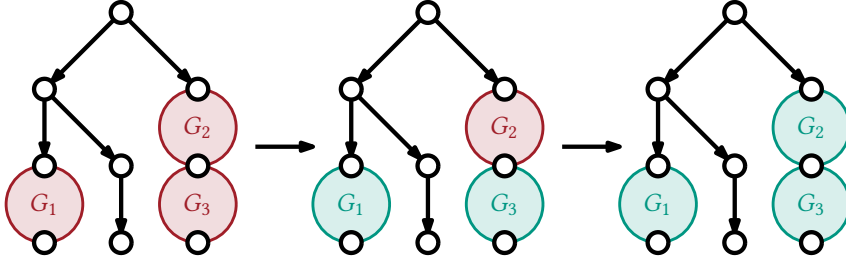


Figure 7.5: Illustration of a Backward Sweep on a flow tree where some edges are replaced by disjoint flow DAGs. The subgraphs are colored red when they are not yet solved in the Backward Sweep and green when they are solved. The process solves these subgraphs in a bottom-up manner and uses the resulting total demands at the roots of the subgraphs to continue this process until it reaches the root of the flow tree.

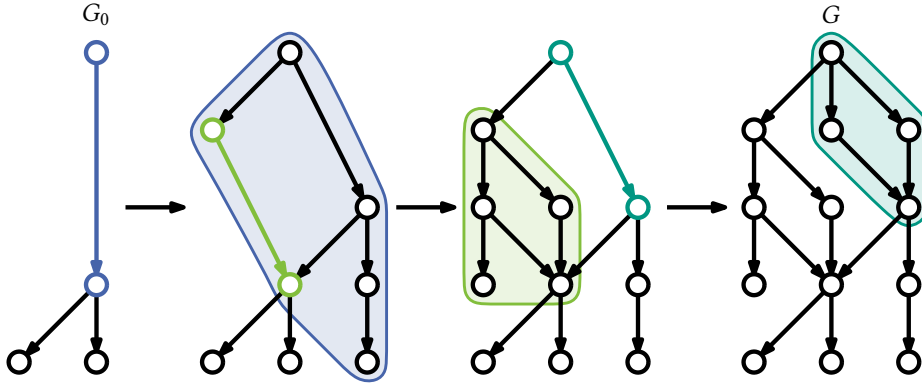


Figure 7.6: Construction of a nested disjoint G . We begin with the flow tree G_0 and replace an edge with a disjoint flow DAG. Then, we replace an edge in that disjoint flow DAG with another disjoint flow DAG twice. Disjoint flow DAGs are highlighted in the same color as the edge they replace.

nested disjoint in the final nested disjoint. It is the inclusion-maximal connected subgraph rooted in u where v is a sink. This is also a nested disjoint, and we denote it as $G_{u,v}$. Intuitively, we can think of $G_{u,v}$ as the nested disjoint that replaces the edge (u, v) by iteratively replacing edges in it with disjoint flow DAGs.

We also provide a non-inductive definition of nested disjoint. Let $G = (V, E, \phi_r, S, w)$ be a flow DAG with counter-monotonic backward and forward functions bw, fw . Every join vertex v_{join} admits a set of ancestors where paths from the root r to v_{join} split and later join again in v_{join} . We call these ancestors *fork vertices* or *forks*, and they are defined as follows. Let u be an ancestor of v_{join} . Vertex u is a fork vertex if there exist at least two disjoint paths from u to v_{join} . Figure 7.7 shows examples of fork vertices.

To use the algorithm from Section 7.4.1, we have to ensure that we can find disjoint flow DAGs in the given flow DAG that we can abstract to a single edge. For this, we demand that every fork vertex u fulfills the following property. If u is a fork vertex of two join vertices v_1 and v_2 , then every pair of paths p_1, p_2 , where p_1 goes from u to v_1 and p_2 goes from u to v_2 must be disjoint. Flow DAGs where every fork vertex satisfies this property are nested

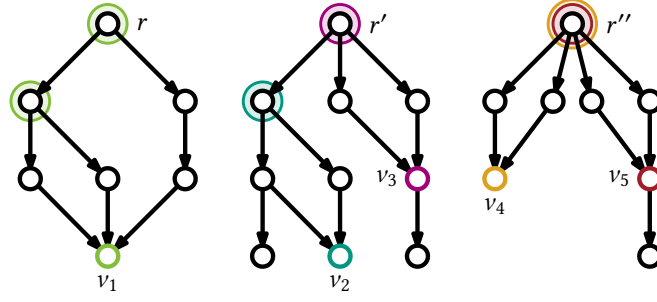


Figure 7.7: Three nested disjoint flow DAGs. Their fork vertices are highlighted in the same color as the corresponding join vertex. Note that the vertices can be fork vertices for multiple join vertices. The first nested disjoint shows that a join vertex like v_1 can have multiple fork vertices. The third nested disjoint is a nested disjoint even though the root is a fork vertex of both v_4 and v_5 since all paths from u to v_4 and u to v_5 are disjoint.

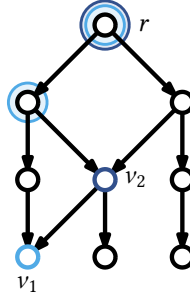


Figure 7.8: A flow DAG that is not a nested disjoint. The root r is a fork vertex for both join vertices v_1 and v_2 . Since there is a path from v_2 to v_1 , this flow DAG is not a nested disjoint.

disjoint flow DAGs. Figure 7.7 shows three examples of nested disjoint flow DAGs. A counterexample of a flow DAG that is not a nested disjoint is shown in Figure 7.8. It should be noted that nested disjoint flow DAGs are flow DAGs where the underlying undirected graph is series-parallel without multi-edges, and the edges are oriented such that the root is the source and the leaves are the sinks. Undirected series-parallel graphs are all graphs with treewidth 2, and we can always direct their edges to obtain a nested disjoint flow DAG.

The first definition of nested disjoint flow DAGs emphasizes how they can be solved like flow trees by solving the disjoint flow DAGs with a modified version of the algorithm from Section 7.4.1 as a subroutine akin to the backward function used for a single edge. Thus, we propose the following algorithm to solve the Flow Problem on a nested disjoint flow DAG G .

7.5.1 Nested Backward-Forward Sweep

Let $G = (V, E, \phi_r, S, w)$ be a nested disjoint flow DAG with counter-monotonic backward and forward functions bw, fw . We modify the Routed Backward-Forward Sweep from Section 7.3 as follows.

We abstract the given nested disjoint G to the original flow tree G_0 from the inductive definition of nested disjoint. In the Backward Sweep, we traverse G_0 in a bottom-up manner. For every edge (u, v) in G_0 that is not replaced by a nested disjoint, we calculate the flow that is sent from u to v as before.

$$bw(\phi_u^{(i)}, \hat{S}_v^{(i+1)})$$

If (u, v) is replaced by a nested disjoint $G_{u,v}$, assume we can solve the Flow Problem on $G_{u,v}$ with our blackbox where we fix the vertex potential $\phi_u^{(i)}$ and simulate the vertex demand at vertex v to be its total demand in G , i.e., $S'_v := \hat{S}_v^{(i+1)}$. Consider a solution to the Flow Problem on $G_{u,v}$ with these fixed values. Then, we can calculate the outgoing flow from u into $G_{u,v}$ as follows. The vertex u has multiple children $v_1, \dots, v_m$ in $G_{u,v}$. We set $\hat{S}_{v_1}^{(i+1)}, \dots, \hat{S}_{v_m}^{(i+1)}$ to be the total vertex demands of the solution to the Flow Problem on $G_{u,v}$. We do this for all vertices of $G_{u,v}$. Then, we can calculate the total flow that is sent from u into $G_{u,v}$ as follows.

$$\sum_{j=1}^m bw(\phi_u^{(i)}, \hat{S}_{v_j}^{(i+1)})$$

This term can be used to replace the term $bw(\phi_u^{(i)}, \hat{S}_v^{(i+1)})$ in the Backward Sweep.

In the Forward Sweep, we traverse G_0 in a top-down manner. For edges (u, v) that are not replaced by a nested disjoint, we calculate the vertex potentials as before.

$$\phi_v^{(i+1)} = fw(\phi_u^{(i+1)}, \hat{S}_v^{(i+1)})$$

Note that in this case, the routing ρ for (u, v) is 1 since no child of u is a join vertex. If (u, v) is replaced by a nested disjoint $G_{u,v}$, we calculated a solution to the Flow Problem on $G_{u,v}$ in the Backward Sweep. We use the vertex potentials of this solution as the vertex potentials of the vertices in $G_{u,v}$ in the next iteration $\phi^{(i+1)}$. Because the vertex potentials and the total vertex demands are a solution to the Flow Problem, the formula

$$\phi_c^{(i+1)} = fw(\phi_p^{(i+1)}, \rho(p, c) \cdot \hat{S}_c^{(i+1)})$$

holds for all vertices $c \neq u$ in $G_{u,v}$ where p is an arbitrary parent of c .

As with the Routed Backward-Forward Sweep, we have to initialize the vertex potentials $\phi^{(0)}$ and repeat this process until the vertex potentials $\phi^{(i)}$ and total vertex demands $\hat{S}^{(i)}$ converge. We call this algorithm the *Nested Backward-Forward Sweep* or *Nested BFS*. As with the Routed Backward-Forward Sweep, we show that if the Nested BFS converges, then the limit vertex potentials $\phi^{(\infty)}$ are a solution to the Flow Problem on G .

Lemma 7.7 (Nested BFS Convergence Solution): *Let G be an instance of the Counter-Monotonic Flow Problem and a nested disjoint flow DAG. If we can solve the Flow Problem on every nested disjoint $G_{u,v}$ encountered in the Nested BFS and the algorithm converges, then the limit vertex potentials $\phi^{(\infty)}$ are a solution of the Flow Problem on G .*

Proof. We show that the limit vertex potentials $\phi^{(\infty)}$ are a solution to the Flow Problem on G . To do this, we show that the limit vertex potentials $\phi^{(\infty)}$ fulfill the following equation for every vertex v in G .

$$S_v = \sum_{u \in p(v)} f_c(\phi_u^{(\infty)}, \phi_v^{(\infty)}) - \sum_{x \in c(v)} f_p(\phi_v^{(\infty)}, \phi_x^{(\infty)})$$

If v is part of a nested disjoint $G_{x,y}$ encountered in the Nested BFS and $v \neq y$ the equation holds since we assumed that we can solve the Flow Problem on $G_{x,y}$. If v is the lowest join vertex of a nested disjoint $G_{x,v}$, we know that we solved the Flow Problem with the simulated vertex demand $S'_v = \hat{S}_v^{(\infty)}$. The solution on $G_{x,v}$ gives us the following.

$$S'_v = \hat{S}_v^{(\infty)} = \sum_{u \in p(v)} f_c(\phi_u^{(\infty)}, \phi_v^{(\infty)})$$

The total vertex demand $\hat{S}_v^{(\infty)}$ is calculated as follows.

$$\begin{aligned} \hat{S}_v^{(\infty)} &= S_v + \sum_{x \in c(v)} bw(\phi_v^{(\infty)}, \rho(v, x) \cdot \hat{S}_x^{(\infty)}) \\ &= S_v + \sum_{x \in c(v)} f_p(\phi_v^{(\infty)}, fw(\phi_v^{(\infty)}, \rho(v, x) \cdot \hat{S}_x^{(\infty)})) \end{aligned}$$

If (v, x) is part of a nested disjoint, we found a solution on that nested disjoint. If (v, x) is not part of a nested disjoint, the following holds by definition of the forward function fw .

$$= S_v + \sum_{x \in c(v)} f_p(\phi_v^{(\infty)}, \phi_x^{(\infty)})$$

Substituting this into the previous equation and rearranging, we get the desired result.

$$S_v = \sum_{u \in p(v)} f_c(\phi_u^{(\infty)}, \phi_v^{(\infty)}) - \sum_{x \in c(v)} f_p(\phi_v^{(\infty)}, \phi_x^{(\infty)})$$

If v is not part of a nested disjoint, the equation holds with the same argument as in Theorem 4.4. This covers all vertices in G and we can conclude that the limit vertex potentials $\phi^{(\infty)}$ are a solution to the Flow Problem on G . $\blacksquare$

It remains to show how we can solve the Flow Problem on $G_{u,v}$ by providing a working blackbox. To do this, we modify the Routing Search from Section 7.4.1 to recursively solve the Flow Problem on the smaller contained nested disjoints.

7.5.2 Nested Routing Search

Let $G_{u,v}$ be a nested disjoint found in the Nested BFS. Because $G_{u,v}$ is not necessarily a disjoint flow DAG, but only a nested disjoint, we have to treat all contained nested disjoints as single edges and solve them recursively with the same approach. For this, we abstract $G_{u,v}$ to a disjoint flow DAG $G'_{u,v}$ which is the original disjoint flow DAG that was used to replace the edge (u, v) in G_0 . We modify the Routing Search as follows. Normally the Routing Search would run the RBFS on the split graph $G_{u,v}^\circ$, but because $G_{u,v}$ is not necessarily a disjoint flow DAG, we run the Nested BFS on the split graph of the abstracted graph $G_{u,v}'^\circ$.

It can happen that a nested disjoint in $G_{u,v}$ contains v as a join vertex. An example of this is shown in Figure 7.6. In this case, the vertex v in $G'_{u,v}$ has fewer parents than in $G_{u,v}$. Therefore, instead of assigning each routing value of the incoming edges of v that were collapsed into a single edge in $G'_{u,v}$, we assign a single routing value and use it as the total routing sum R_1 in the Routing Search for the contained nested disjoint. We call this algorithm the *Nested Routing Search*.

We now show that the Nested Routing Search finds a solution of the Flow Problem on a nested disjoint $G_{u,v}$ if such a solution exists.

Theorem 7.8 (Nested Routing Search Correctness): *Let $G_{u,v} \subseteq G$ be a nested disjoint flow DAG found in the Nested BFS on an instance G of the Counter-Monotonic Flow Problem. Let all Nested BFSs converge. If and only if a solution to the Flow Problem on $G_{u,v}$ exists, then the Nested Routing Search finds it.*

Proof. We prove this by structural induction on $G_{u,v}$. If no edge in $G_{u,v}$ is replaced by a nested disjoint to form G , then $G_{u,v}$ is a disjoint flow DAG and the Nested Routing Search is equivalent to the Routing Search. By Theorem 7.6, the Routing Search finds a solution to the Flow Problem on $G_{u,v}$ if and only if such a solution exists. Now, assume that the Nested Routing Search finds a solution on every nested disjoint $G_{u',v'} \neq G_{u,v}$ that is contained in $G_{u,v}$. By Lemma 7.7, the Nested BFS converges to a solution to the Flow Problem on $G_{u,v}^\circ$. Lemma 7.2 and Lemma 7.3 still hold for the Nested BFS. Therefore, we can apply the same binary search as in the Routing Search to solve the Flow Problem on $G_{u,v}$. ■

With this theorem and the previous results, we can conclude that if the Nested BFS converges, it finds a solution to the Flow Problem on any nested disjoint G . We sum this up in a corollary.

Corollary 7.9 (Nested BFS Correctness): *Let G be an instance of the Counter-Monotonic Flow Problem and a nested disjoint flow DAG. If the Nested BFS converges, then the limit vertex potentials $\phi^{(\infty)}$ are a solution of the Flow Problem on G .*

Proof. This follows directly from Lemma 7.7 and Theorem 7.8. ■

When working with the Modified Power Flow functions of Section 6.1, we can guarantee convergence of the Nested BFS if a solution exists. This follows from Corollary 6.8 and Theorem 7.9.

8 Evaluation

In this chapter, we evaluate the theoretical results for the Backward-Forward Sweep on the Flow Problem with Modified Power Flow functions. We empirically investigate the convergence behavior of the BFS and analyze how it is affected by the structure of the flow tree, the vertex demands, the root potential, and the convergence precision.

To conduct these experiments, we first describe methods for generating feasible instances of the Flow Problem. We then focus on palm trees introduced in Section 6.2 for which we derived an upper bound for the demand. In particular, we study the influence of the demand, the height and width of the palm tree, the root potential, and the convergence threshold on the number of iterations required for ε -convergence. Finally, we evaluate the Routing Search and the Nested Backward-Forward Sweep for small instances of disjoint flow DAGs and nested disjoint flow DAGs, respectively, and analyze how they are affected by the structure of the graph.

8.1 Generating Feasible Instances

To test and analyze solution algorithms like the BFS for the Flow Problem with Modified Power Flow functions, it is useful to generate feasible instances of the Flow Problem. When generating such instances, one might try to set the vertex demands S directly and then compute the vertex potentials ϕ of a solution. We will not focus on this approach, since the connection between vertex demands and the feasibility of the instance is not straightforward. This leads to the problem that we might generate vertex demands S for which no feasible solution exists. Instead, we can generate the vertex potentials ϕ and then compute the supplied vertex demands S . This section describes different methods to generate such vertex potentials.

If we want to have a fixed root potential ϕ_r , we can generate the vertex potentials in a top-down manner while guaranteeing that the resulting vertex demands S are non-negative. We will see that some resulting instances are harder to solve than others depending on the choice of the vertex potentials.

If we do not care about the root potential ϕ_r , we can generate the vertex potentials in a bottom-up manner while setting the leaf potentials as we choose.

8.1.1 Top-Down-Generation

If we want to guarantee an upper bound for the vertex potentials, we can fix the root vertex potential and generate the vertex potentials in a top-down manner. Let the root vertex r have a potential $\phi_r \geq 0$. For every child x of r , we can uniformly choose arbitrary potentials $0 \leq \phi_x \leq \phi_r$, i.e., $\phi_x \sim \mathcal{U}([0, \phi_r])$. We can generate the vertex potentials ϕ_v for every vertex

v in a top-down manner. Let v be a non-root vertex with parent u and edge $e = (u, v)$. Let ϕ_u, ϕ_v be known vertex potentials. We generate the vertex potentials ϕ_x for every child x of v as follows. For the demand S_v to be non-negative, we must satisfy the following condition.

$$\begin{aligned} 0 &\leq \sum_{u \in p(v)} f_c(\phi_u, \phi_v) - \sum_{x \in c(v)} f_p(\phi_v, \phi_x) \\ \Leftrightarrow \quad 0 &\leq \sum_{u \in p(v)} (\phi_u - \phi_v) \cdot \phi_v - \sum_{x \in c(v)} (\phi_v - \phi_x) \cdot \phi_v \end{aligned}$$

Dividing both sides by ϕ_v , which is non-negative by assumption, we get the following.

$$\Leftrightarrow \quad 0 \leq \sum_{u \in p(v)} (\phi_u - \phi_v) - \sum_{x \in c(v)} (\phi_v - \phi_x)$$

Rearranging the terms gives us the following.

$$\Leftrightarrow \quad \sum_{x \in c(v)} \phi_x \geq (|c(v)| + |p(v)|) \cdot \phi_v - \sum_{u \in p(v)} \phi_u$$

To generate the vertex potentials for a child x of v uniformly at random, we can conclude that

$$\phi_x \geq \frac{(|c(v)| + |p(v)|) \cdot \phi_v - \sum_{u \in p(v)} \phi_u}{|c(v)|}$$

must hold for every child x of v . Since the right-hand side can be negative, we define the lower bound $\phi_{\min}$ for the vertex potentials of the children of v as follows.

$$\phi_{\min} := \max \left\{ 0, \frac{(|c(v)| + |p(v)|) \cdot \phi_v - \sum_{u \in p(v)} \phi_u}{|c(v)|} \right\}$$

We sample the vertex potentials ϕ_x for every child x of v uniformly at random from the interval $[\phi_{\min}, \phi_v]$, i.e., $\phi_x \sim \mathcal{U}([\phi_{\min}, \phi_v])$. To show that this interval is well-defined, we have to show that $\phi_{\min} \leq \phi_v$. If $\phi_{\min} = 0$, this is true by our assumption that $\phi_v \geq 0$. We now assume the other case and use the fact that $\phi_u \geq \phi_v$ for all $u \in p(v)$ as by Observation 6.5.

$$\phi_{\min} = \frac{(|c(v)| + |p(v)|) \cdot \phi_v - \sum_{u \in p(v)} \phi_u}{|c(v)|} \leq \frac{(|c(v)| + |p(v)|) \cdot \phi_v - |p(v)| \cdot \phi_v}{|c(v)|} = \phi_v$$

Note that this method does not generate all feasible instances on a given flow graph, since we sample the vertex potentials for all children of a vertex v from the same interval $[\phi_{\min}, \phi_v]$. This excludes cases where the vertex potentials are heavily unbalanced, i.e., some children have very low vertex potentials while others have very high vertex potentials.

8.1.2 Epsilon-Top-Down-Generation

The last section describes a method to generate vertex potentials in a top-down manner, but does not give us much control over the magnitude and spacing of the generated vertex potentials. In this section, we modify the method from Section 8.1.1 to give us more control over the generated vertex potentials. Let the generation of the vertex potentials be as in

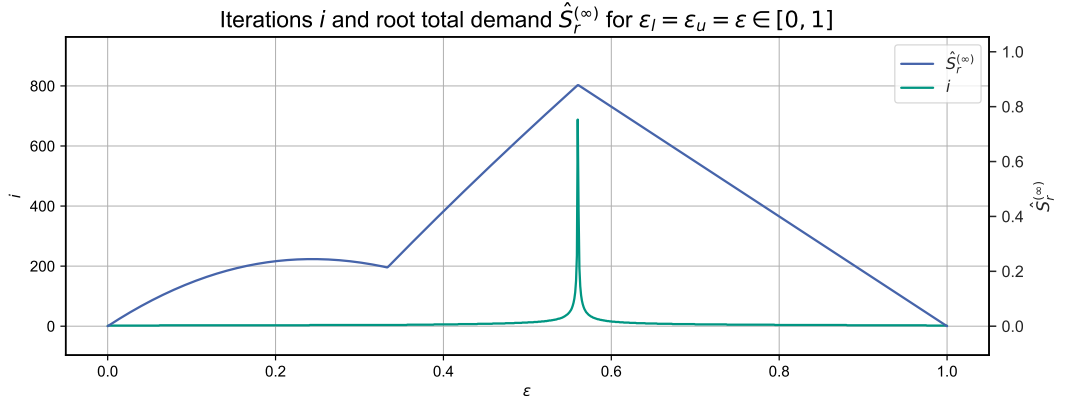


Figure 8.1: BFS-iterations i until 10^{-8} -convergence and the solution total demand of the root vertex $\hat{S}_r^{(\infty)}$ as a function of the sampling parameter $\varepsilon = \varepsilon_l = \varepsilon_u$ for a complete binary tree of height 8. The root vertex potential is set to $\phi_r = 1$. The left y -axis shows the number of BFS-iterations, and the right y -axis shows the solution total demand.

Section 8.1.1, but we modify the intervals from which we sample the vertex potentials. We define two parameters $\varepsilon_l, \varepsilon_u \in [0, 1]$ with $\varepsilon_l \leq \varepsilon_u$. We now sample between from the ε_l and ε_u fractions of the original interval. That is, for each child x of v , we sample its potential uniformly from the following interval.

$$[\phi_{\min}, \phi_v]_{\varepsilon_l}^{\varepsilon_u} := [\phi_{\min} + (\phi_v - \phi_{\min}) \cdot \varepsilon_l, \phi_{\min} + (\phi_v - \phi_{\min}) \cdot \varepsilon_u]$$

For the children x of the root r , we sample their potentials uniformly from the following interval.

$$[0, \phi_r]_{\varepsilon_l}^{\varepsilon_u} := [\phi_r \cdot \varepsilon_l, \phi_r \cdot \varepsilon_u]$$

Varying the values of ε_l and ε_u gives us more control over the magnitude of the generated vertex potentials. For example, sampling with a high value of ε_l will result in higher average value of the vertex potentials.

We begin by simplifying this and always setting $\varepsilon_l = \varepsilon_u$ and observe the impact of this parameter on the ε -convergence speed of the BFS.

Figure 8.1 shows the number of iterations i until convergence and the solution total demand of the root vertex $\hat{S}_r^{(\infty)}$ as a function of the sampling parameter $\varepsilon = \varepsilon_l = \varepsilon_u$ for a fixed flow graph. We can see that the number of iterations i are maximal for $\varepsilon \approx 0.56$. This is consistent with the maximum of the roots total demand $\hat{S}_r^{(\infty)}$ of the solution and indicates that the BFS converges slower for instances with higher total demand.

To support this observation, we now vary both parameters ε_l and ε_u independently and observe the impact on convergence speed of the BFS. Figure 8.2 shows the number of iterations i until convergence and the solution total demand of the root vertex $\hat{S}_r^{(\infty)}$ as a function of the sampling parameters ε_l and ε_u for a fixed flow graph. Again, we can see that the number of iterations i are maximal when the solution total demand of the root vertex $\hat{S}_r^{(\infty)}$ is maximal.

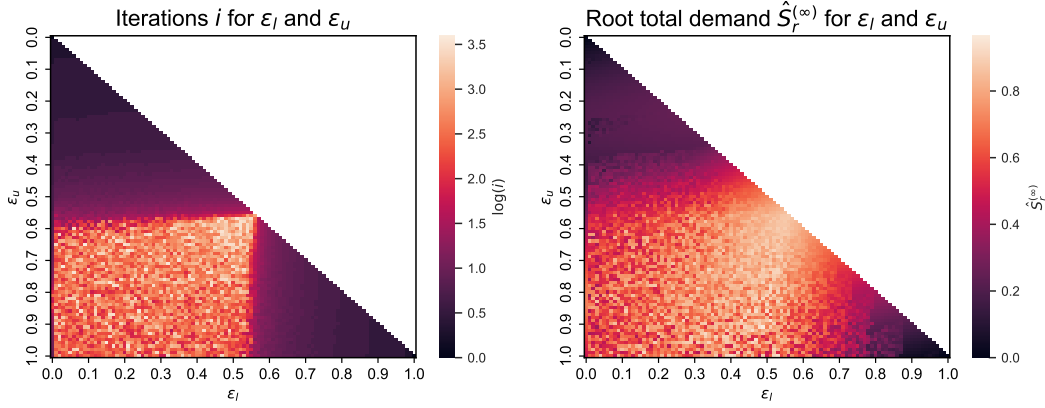


Figure 8.2: The left figure shows a heatmap of the maximum BFS-iterations i over 100 runs until 10^{-8} -convergence as a function of the sampling parameters ε_l and ε_u for a complete binary tree of height 8. The heatmap is scaled logarithmically. The right graphic shows a heatmap of the solution total demand of the root vertex $\hat{S}_r^{(\infty)}$ for the corresponding run in the left graphic as a function of the sampling parameters ε_l and ε_u for a complete binary tree of height 8.

8.1.3 Bottom-Up-Generation

While generating potentials top-down guarantees an upper bound for the potential of the root vertex, we can also focus on fixing the leaf potentials. For that, we begin by setting the leaf potentials as we choose. Then, we can generate the vertex potentials for the parent vertices in a bottom-up manner. From the last section, we know that for a vertex v , the following condition must hold for the demand S_v to be non-negative.

$$\sum_{u \in p(v)} \phi_u \geq (|c(v)| + |p(v)|) \cdot \phi_v - \sum_{x \in c(v)} \phi_x$$

To generate the potential of u uniformly at random, we divide both sides by $|p(v)|$ and get the following lower bound for the potential of u .

$$\phi_u \geq \frac{(|c(v)| + |p(v)|) \cdot \phi_v - \sum_{x \in c(v)} \phi_x}{|p(v)|}$$

Among all children v of u , we take the maximum of these lower bounds to guarantee that the resulting vertex demands S are non-negative. Using this, we can define a lower bound $\phi_{\min}$ for the potential of u .

$$\phi_{\min} := \max_{v \in c(u)} \left\{ \frac{(|c(v)| + |p(v)|) \cdot \phi_v - \sum_{x \in c(v)} \phi_x}{|p(v)|} \right\}$$

We can now sample the potential of u uniformly at random from the interval $[\phi_{\min}, \infty]$. This process can be repeated until all vertex potentials are generated.

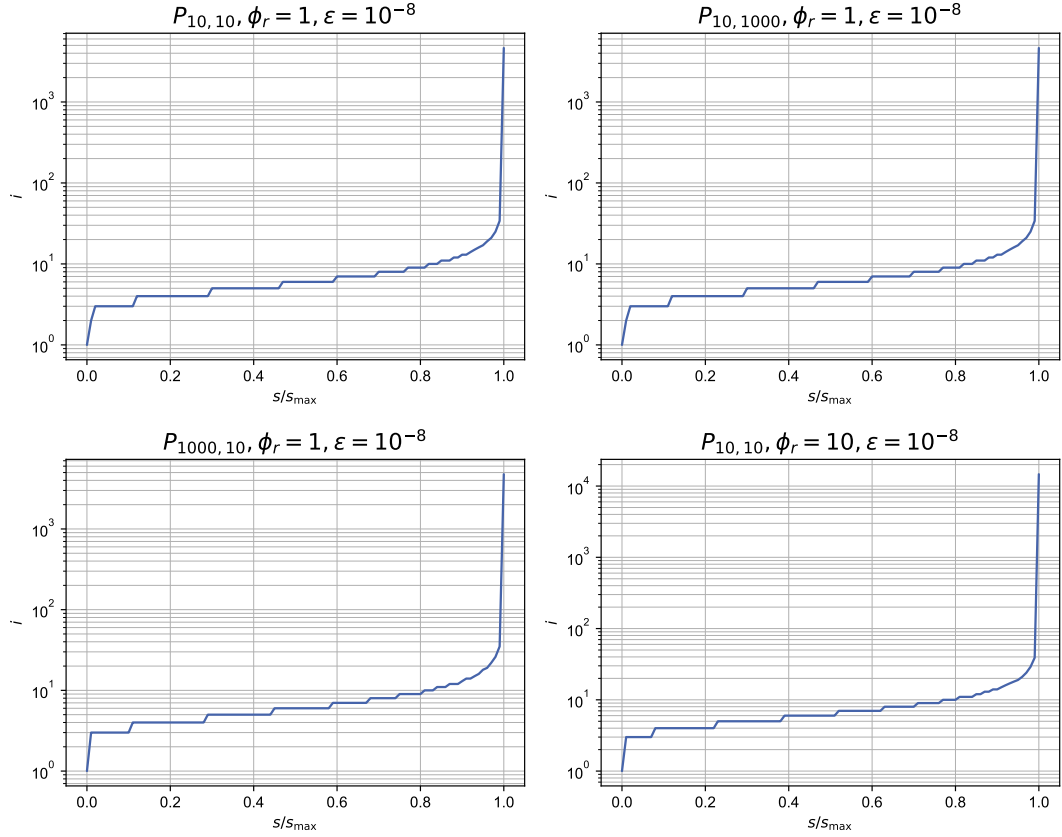


Figure 8.3: BFS-iterations i until ε -convergence as a function of the demand s for four different palm trees $P_{m,n}$ and root vertex potentials ϕ_r . The x -axes are scaled to $s_{\max}$ for each palm tree. The y -axes are scaled logarithmically.

8.2 Palm Trees

In Section 6.2, we introduced palm trees, which are a special class of flow trees composed of a path with m vertices and n leaves attached to the last vertex of the path. We derived an upper bound for the demand s that can be satisfied at the leaves of a palm tree $P_{m,n}$ with root vertex potential ϕ_r in Theorem 6.10. With this upper bound for s , we can generate feasible instances of the Flow Problem with Modified Power Flow functions on palm trees $P_{m,n}$. First, we analyze the convergence speed of the BFS on these instances by measuring the iterations i of the BFS until ε -convergence for different values of $s \leq s_{\max}$ while varying the height m , width n and root vertex potential ϕ_r of the palm tree $P_{m,n}$.

As we can see in Figure 8.3, the number of iterations i until ε -convergence increases as s approaches $s_{\max}$. When comparing the different plots, we can see that the variation of the height m and width n does not change the ε -convergence speed of the BFS significantly. Oppositely, the variation of the root vertex potential ϕ_r has a significant impact on the ε -convergence speed of the BFS such that the number of iterations i until ε -convergence increases as ϕ_r increases. When comparing the first and last plot, we can see that the number of iterations i almost increases by a factor of 10 when increasing the root vertex potential ϕ_r from 1 to 10. A notable observation is that the shape of all curves is very similar, which

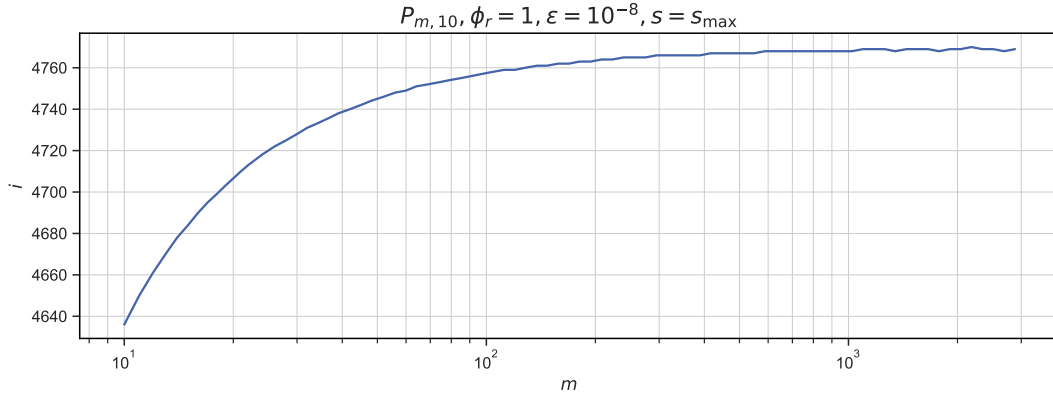


Figure 8.4: BFS-iterations i until ϵ -convergence as a function of the height m of the palm tree. The demand s is set to $s_{\max}$ for each palm tree. The x -axis is scaled logarithmically.

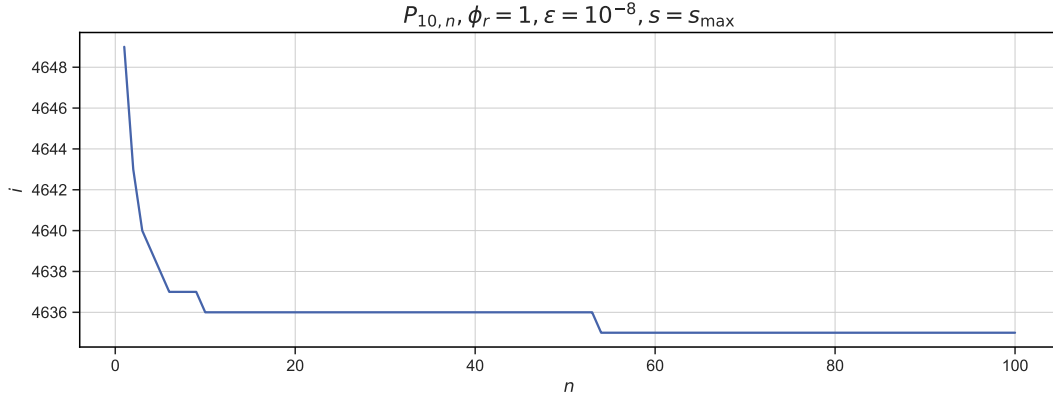


Figure 8.5: BFS-iterations i until ϵ -convergence as a function of the width n of the palm tree. The demand s is set to $s_{\max}$ for each palm tree.

indicates that the ϵ -convergence speed of the BFS is mainly determined by the ratio $s/s_{\max}$ and not by the absolute value of s , since $s_{\max}$ is different for each shown palm tree. Because the y -axes are scaled logarithmically, we can conclude that the ϵ -convergence speed of the BFS is just scaled between different palm trees by some factor independent of the ratio $s/s_{\max}$.

To see the impact of m and n more clearly, we now fix all parameters of the palm tree $P_{m,n}$ except for m and n and vary them. As seen in Figure 8.4, the number of iterations i until ϵ -convergence increases very slowly as m increases. Figure 8.5 shows that the number of iterations i until ϵ -convergence only decreases as n increases. This means, we can approximate the impact of n on the ϵ -convergence speed of the BFS by a constant factor.

In the previous analysis, we fixed the value of ϵ to 10^{-8} . Now, we vary the value of ϵ and observe its impact on the ϵ -convergence speed of the BFS. Doing this, the number of iterations i until ϵ -convergence increases as ϵ decreases. This is intuitive, since the BFS is executed equally for all values of ϵ but stops earlier for larger values of ϵ . We can quantify the impact of ϵ . When measuring the iterations i until ϵ -convergence for different values of ϵ , we observe a linear relationship between the number of iterations i until ϵ -convergence and the value of

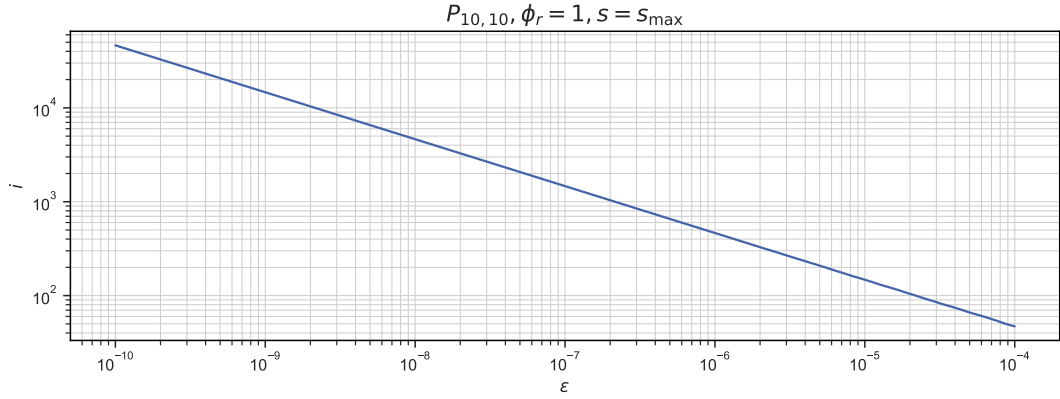


Figure 8.6: BFS-iterations i until ϵ -convergence as a function of ϵ . The demand s is set to $s_{\max}$ for each palm tree. Both axes are scaled logarithmically.

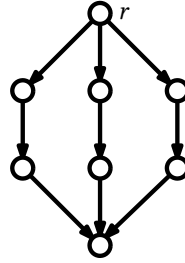


Figure 8.7: A disjoint flow DAG with $w = 3$ and $h = 2$. Note that h describes the number of vertices between root and join on each path, not the number of edges.

ϵ on a logarithmic scale. The slope of this linear relationship is $-\frac{1}{2}$, which indicates that the number of iterations i until ϵ -convergence is inversely proportional to $\sqrt{\epsilon}$. This is shown in Figure 8.6.

8.3 Routing Search

In Section 7.4.1, we introduced the Routing Search to find a correct routing for a given disjoint flow DAG. Now, we evaluate the Routing Search empirically and analyze how it is affected by the structure of the graph. For this, we use disjoint flow DAGs made up of parallel paths of equal length, which are joined at the source and sink vertices. This gives us two parameters to vary, the number of parallel paths w and the length of the paths h . An example for such a disjoint flow DAG is shown in Figure 8.7.

When analyzing the Routing Search on these disjoint flow DAGs, we can measure two quantities. First, the amount of times i_{bfs} the BFS is executed during the Routing Search. Second, the sum of the number of BFS iterations over all executions during the Routing Search. We denote this quantity as i_{total} .

As we can see in Figure 8.8, the number of times i_{bfs} the BFS is executed during the Routing Search only increases minimally as the path length h increases. We suspect that this increase is due to the relationship between ϵ_{bin} and ϵ_{bfs} , as stated in Section 7.4.1.

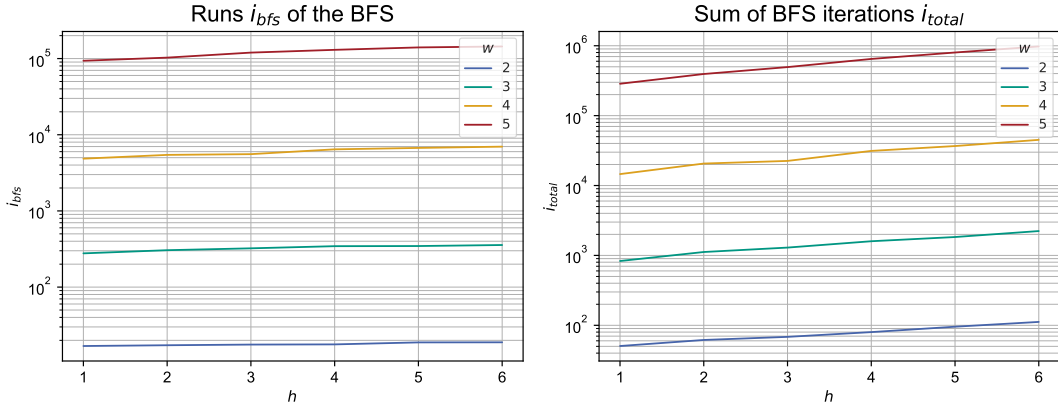


Figure 8.8: The left figure shows the number of times i_{bfs} the BFS is executed during the Routing Search as a function of the path length h . This is shown for different numbers of parallel paths w . The right figure shows the sum of the number of BFS iterations i_{total} over all executions during the Routing Search as a function of the path length h . This is shown for different numbers of parallel paths w . Both y -axes are scaled logarithmically and the values of i_{bfs} and i_{total} are an average over 20 runs. The vertex potentials are generated using the method described in Section 8.1.1 with root potential $\phi_r = 1$. The binary search precision ε_{bin} is set to 10^{-10} and the BFS precision ε_{bfs} is set to 10^{-7} .

On the other hand, the sum of the number of BFS iterations i_{total} over all executions during the Routing Search increases more significantly as the path length h increases. This is intuitive, since the BFS takes more iterations to converge for longer paths. Both plots show that the number of parallel paths w has a much more significant impact on the values. Between the different values of w , the values of i_{bfs} and i_{total} increase by approximately a factor of 10 for each additional parallel path. This is consistent with the theoretical results in Section 7.4.1, where the number of times i_{bfs} the BFS is executed during the Routing Search lies in $\mathcal{O}(\log^{w-1}(\frac{1}{\varepsilon_{bin}}))$.

8.4 Nested Backward-Forward Sweep

We proposed the Nested Backward-Forward Sweep (NBFS) as a generalization of the BFS for nested disjoint flow DAGs in Section 7.5.1. As a proof of concept, we implemented the NBFS and now evaluate it empirically and analyze how it is affected by the structure of the graph.

For this, we introduce two families of nested disjoint. We define recursively nested disjoint N_i and series nested disjoint S_i as shown in Figure 8.9. We do this to analyze the impact of the number of nested levels and contained nested disjoint on the number of iterations i_{total} of the NBFS.

Figure 8.10 shows that the number of iterations i_{total} of the NBFS increases exponentially as the number of nested levels k increases for recursively nested disjoint N_i . This is to be expected, since solving N_i requires solving N_{i-1} in every iteration of the NBFS. On the other hand, the number of iterations i_{total} of the NBFS increases approximately linearly as the number of contained nested disjoint k increases for series nested disjoint S_i . This is the expected behavior, since solving S_i involves solving S_0 i times in each iteration of the NBFS.

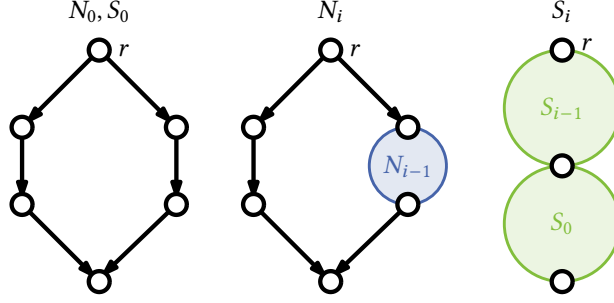


Figure 8.9: The left graph shows the base case for both recursively and series nested disjoint N_0 and S_0 . The center graph shows the recursively nested disjoint N_i where an edge has been replaced by a recursively nested disjoint N_{i-1} . The right graph shows the series nested disjoint S_i as a series of S_{i-1} and S_0 .

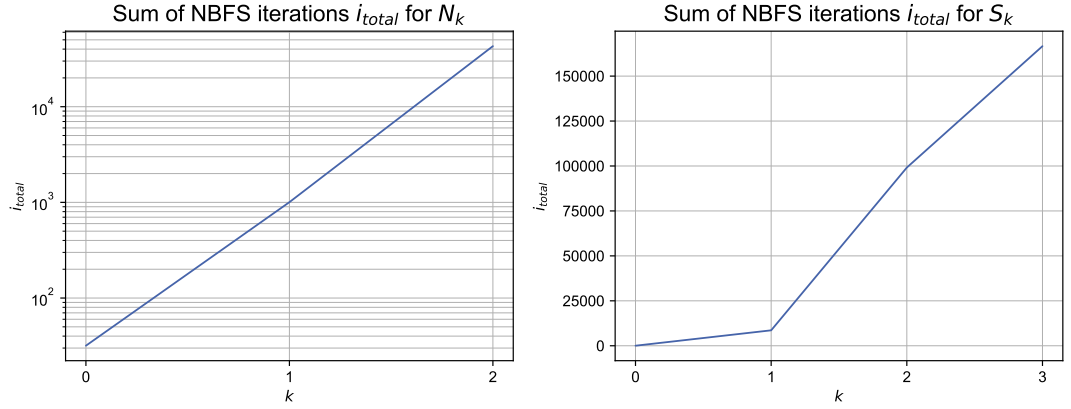


Figure 8.10: The left figure shows the sum of NBFS iterations i_{total} over all executions during the NBFS as a function of the number of nested levels i for recursively nested disjoint N_i . The right figure shows i_{total} as a function of the number of contained nested disjoint i for series nested disjoint S_i . The total iterations i_{total} contain all iterations of all NBFS executions during the NBFS, including the executions of the NBFS in the Routing Search. The left y -axis is scaled logarithmically and the values of i_{total} are an average over 20 runs. The vertex potentials are generated using the method described in Section 8.1.1 with root potential $\phi_r = 1$. The binary search precision ϵ_{bin} is set to 10^{-10} and the BFS precision ϵ_{bfs} is set to 10^{-7} .

Note that we only evaluated the NBFS for a very small number of nested levels and contained nested disjoints, since the number of iterations i_{total} increases very quickly and becomes computationally expensive.

9 Conclusion

This thesis studied the Backward-Forward Sweep (BFS) as a general algorithmic framework for solving flow problems on networks. We abstracted from the electrical interpretation of the BFS and formalized the notations of flow functions, forward and backward functions, vertex potentials, and total vertex demands.

For flow trees, we showed that the BFS computes a solution whenever its vertex potentials converge. We further considered two classes of flow functions. For potential-independent backward functions, the total vertex demands do not depend on the current vertex potentials, and the BFS finds a solution after a single iteration. For counter-monotonic backward and forward functions, we established monotonicity properties of the vertex potentials and total vertex demands. These properties guarantee that the BFS converges if the initial vertex potentials undersupply the vertex demands and the vertex potentials are bounded below.

We applied this framework to the Modified Power Flow functions, which are a simplification of the AC power flow equations used in practice. Their backward and forward functions are counter-monotonic. Consequently, when a non-negative solution exists and the vertex potentials are initialized with the root potential, the BFS converges to a solution. If no such solution exists, the algorithm detects this through an undefined backward or forward function. The analysis of palm trees further illustrated how the structure and demands of an instance influence feasibility and convergence.

We also extended the Flow Problem from directed trees to directed acyclic graphs. Since vertices in a directed acyclic graph can have multiple parents, we introduced routing functions and split graphs. The resulting RBFS solves the problem on the split graph when a correct routing is given. For disjoint flow DAGs, we developed a Routing Search that uses the monotonicity of the vertex potentials to find a correct routing. Finally, we combined these ideas recursively in the Nested BFS and showed how it can solve Flow Problems on nested disjoint flow DAGs. These are directed acyclic graphs whose underlying undirected graphs have treewidth at most 2.

The results demonstrate that the BFS is not limited to its traditional application in radial electrical networks. Its framework can be studied independently of a specific physical flow model and can be extended to more general network topologies. At the same time, the convergence of the BFS depends on the concrete choice of flow functions, initialization, and feasibility of the instance.

9.1 Future Work

Several questions remain open. First, the convergence results could be strengthened by deriving explicit convergence rates and by identifying weaker conditions than counter-monotonicity. In particular, it would be valuable to determine an upper bound for the number of iterations required for the BFS to converge depending on the instance parameters. In Section 8.2 we observed that number of iterations is mainly influenced by the vertex demands of the instance. Although we only considered Palm Trees, we suspect that the main factor

influencing the convergence speed on general graphs are the total vertex demands of the solution. Based on this, it would be interesting to study the convergence speed of the BFS in relation to the magnitude and distribution of vertex demands, e.g., if the demands are concentrated near the root or the leaves.

Second, we only considered a subclass of directed acyclic graphs where the routing for every join vertex can be found independently of other join vertices. General directed acyclic graphs require a more sophisticated routing search that considers the interactions between join vertices.

Third, the relationship between the precision of the binary search and the precision of the BFS could be studied in more detail. This would help in understanding how the choice of precision affects the overall performance and convergence of the algorithm.

Finally, the practical performance of the Nested BFS could be improved. At the moment, the algorithm requires solving the Flow Problem for complete contained nested disjoints. This is not necessary in every case as solving the Flow Problem on just the paths from fork vertices to the leaves while aggregating the total demands of other parts beforehand is sufficient. This reduces the number of subgraphs nested in other subgraphs and therefore the number of times the Flow Problem must be solved. Also, when binary searching the routing in the Routing Search algorithm many values are calculated multiple times. This could be improved by eliminating routings that are known to be incorrect and reusing previously computed values.

Bibliography

- [AMS26] Ayesha, Gabriele Mosaico, and Federico Silvestro. “Power Flow Methods for Efficient Analysis of Modern Distribution Networks—Review”. In: *Energies* Volume 19 (2026). ISSN: 1996-1073. DOI: [10.3390/en19163902](https://doi.org/10.3390/en19163902).
- [BCCN00] E. Bompard, E. Carpaneto, G. Chicco, and R. Napoli. “Convergence of the backward/forward sweep method for the load-flow analysis of radial distribution systems”. In: *International Journal of Electrical Power & Energy Systems* Volume 22 (2000), pp. 521–530. ISSN: 0142-0615. DOI: [https://doi.org/10.1016/S0142-0615\(00\)00009-0](https://doi.org/10.1016/S0142-0615(00)00009-0).
- [BCHP63] H. E. Brown, G. K. Carter, H. H. Happ, and C. E. Person. “Power Flow Solution by Impedance Matrix Iterative Method”. In: *IEEE Transactions on Power Apparatus and Systems* Volume 82 (1963), pp. 1–10. DOI: [10.1109/TPAS.1963.291392](https://doi.org/10.1109/TPAS.1963.291392).
- [BV19] Daniel Bienstock and Abhinav Verma. “Strong NP-hardness of AC power flows feasibility”. In: *Operations Research Letters* Volume 47 (2019), pp. 494–501. ISSN: 0167-6377. DOI: <https://doi.org/10.1016/j.orl.2019.08.009>.
- [Cap16] Florin Capitanescu. “Critical review of recent advances and further developments needed in AC optimal power flow”. In: *Electric Power Systems Research* Volume 136 (2016), pp. 57–68. ISSN: 0378-7796. DOI: <https://doi.org/10.1016/j.epsr.2016.02.008>.
- [Che+91] T.-H. Chen, M.-S. Chen, K.-J. Hwang, P. Kotas, and E.A. Chebli. “Distribution system power flow analysis—a rigid approach”. In: *IEEE Transactions on Power Delivery* Volume 6 (1991), pp. 1146–1152. DOI: [10.1109/61.85860](https://doi.org/10.1109/61.85860).
- [CS95] C.S. Cheng and D. Shirmohammadi. “A three-phase power flow method for real-time distribution system analysis”. In: *IEEE Transactions on Power Systems* Volume 10 (1995), pp. 671–679. DOI: [10.1109/59.387902](https://doi.org/10.1109/59.387902).
- [DC13] Jiao-Jiao Deng and Hsiao-Dong Chiang. “Convergence Region of Newton Iterative Power Flow Method: Numerical Studies”. In: *Journal of Applied Mathematics* Volume 2013 (2013), p. 509496. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1155/2013/509496>.
- [DKK95] D. Das, D.P. Kothari, and A. Kalam. “Simple and efficient method for load flow solution of radial distribution networks”. In: *International Journal of Electrical Power & Energy Systems* Volume 17 (1995), pp. 335–346. ISSN: 0142-0615. DOI: [https://doi.org/10.1016/0142-0615\(95\)00050-0](https://doi.org/10.1016/0142-0615(95)00050-0).
- [dPdB18] Leandro Ramos de Araujo, Débora Rosana Ribeiro Penido, Nélío Alves do Amaral Filho, and Tatianna Aparecida Pereira Beneteli. “Sensitivity analysis of convergence characteristics in power flow methods for distribution systems”. In: *International Journal of Electrical Power & Energy Systems* Volume 97 (2018), pp. 211–219. ISSN: 0142-0615. DOI: <https://doi.org/10.1016/j.ijepes.2017.11.012>.

- [DVE09] A. Dukpa, B. Venkatesh, and M. El-Hawary. “Application of continuation power flow method in radial distribution systems”. In: *Electric Power Systems Research* Volume 79 (2009), pp. 1503–1510. ISSN: 0378-7796. DOI: <https://doi.org/10.1016/j.epsr.2009.05.003>.
- [EH08] U. Eminoglu and M. H. Hocaoglu. “Distribution Systems Forward/Backward Sweep-based Power Flow Algorithms: A Review and Comparison Study”. In: *Electric Power Components and Systems* Volume 37 (2008), pp. 91–110. eprint: <https://doi.org/10.1080/15325000802322046>.
- [FN23] Francisco Damasceno Freitas and Laice Neves de Oliveira. “Conditioning step on the initial estimate when solving ill-conditioned power flow problems”. In: *International Journal of Electrical Power & Energy Systems* Volume 146 (2023), p. 108772. ISSN: 0142-0615. DOI: <https://doi.org/10.1016/j.ijepes.2022.108772>.
- [FZL25] Bohang Fang, Changhong Zhao, and Steven H. Low. “Convergence of Backward/Forward Sweep for Power Flow Solution in Radial Networks”. In: *IEEE Transactions on Control of Network Systems* Volume 12 (2025), pp. 1780–1792. DOI: [10.1109/TCNS.2025.3534557](https://doi.org/10.1109/TCNS.2025.3534557).
- [GB91] S.K. Goswami and S.K. Basu. “Direct solution of distribution systems”. In: *IEE Proceedings C (Generation, Transmission and Distribution)* Volume 138 (1991), pp. 78–88. eprint: <https://digital-library.theiet.org/doi/pdf/10.1049/ip-c.1991.0010>.
- [GR99] Antonio Gómez Expósito and E. Romero Ramos. “Reliable load flow technique for radial distribution networks”. In: *IEEE Transactions on Power Systems* Volume 14 (1999), pp. 1063–1069. DOI: [10.1109/59.780924](https://doi.org/10.1109/59.780924).
- [Ker91] W.H. Kersting. “Radial distribution test feeders”. In: *IEEE Transactions on Power Systems* Volume 6 (1991), pp. 975–985. DOI: [10.1109/59.119237](https://doi.org/10.1109/59.119237).
- [LGV16] Karsten Lehmann, Alban Grastien, and Pascal Van Hentenryck. “AC-Feasibility on Tree Networks is NP-Hard”. In: *IEEE Transactions on Power Systems* Volume 31 (2016), pp. 798–801. DOI: [10.1109/TPWRS.2015.2407363](https://doi.org/10.1109/TPWRS.2015.2407363).
- [LS90] G.X. Luo and A. Semlyen. “Efficient load flow for large weakly meshed networks”. In: *IEEE Transactions on Power Systems* Volume 5 (1990), pp. 1309–1316. DOI: [10.1109/59.99382](https://doi.org/10.1109/59.99382).
- [MMH21] Oscar Danilo Montoya, Alexander Molina-Cabrera, and Jesus C. Hernández. “A Comparative Study on Power Flow Methods Applied to AC Distribution Networks with Single-Phase Representation”. In: *Electronics* Volume 10 (2021). ISSN: 2079-9292. DOI: [10.3390/electronics10212573](https://doi.org/10.3390/electronics10212573).
- [SA74] B. Stott and O. Alsac. “Fast Decoupled Load Flow”. In: *IEEE Transactions on Power Apparatus and Systems* Volume PAS-93 (1974), pp. 859–869. DOI: [10.1109/TPAS.1974.293985](https://doi.org/10.1109/TPAS.1974.293985).
- [SHSL88] D. Shirmohammadi, H.W. Hong, A. Semlyen, and G.X. Luo. “A compensation-based power flow method for weakly meshed distribution and transmission networks”. In: *IEEE Transactions on Power Systems* Volume 3 (1988), pp. 753–762. DOI: [10.1109/59.192932](https://doi.org/10.1109/59.192932).
- [TH67] William. F. Tinney and Clifford E. Hart. “Power Flow Solution by Newton’s Method”. In: *IEEE Transactions on Power Apparatus and Systems* Volume PAS-86 (1967), pp. 1449–1460. DOI: [10.1109/TPAS.1967.291823](https://doi.org/10.1109/TPAS.1967.291823).

-
- [TWJ99] D. Thukaram, H.M. Wijekoon Banda, and Jovitha Jerome. “A robust three phase power flow algorithm for radial distribution systems”. In: *Electric Power Systems Research* Volume 50 (1999), pp. 227–236. ISSN: 0378-7796. DOI: [https://doi.org/10.1016/S0378-7796\(98\)00150-3](https://doi.org/10.1016/S0378-7796(98)00150-3).
- [VG61] James E. Van Ness and John H. Griffin. “Elimination Methods for Load-Flow Studies”. In: *Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems* Volume 80 (1961), pp. 299–302. DOI: [10.1109/AIEEPAS.1961.4501030](https://doi.org/10.1109/AIEEPAS.1961.4501030).
- [WH56] J. B. Ward and H. W. Hale. “Digital Computer Solution of Power-Flow Problems [includes discussion]”. In: *Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems* Volume 75 (1956), pp. 398–404. DOI: [10.1109/AIEEPAS.1956.4499318](https://doi.org/10.1109/AIEEPAS.1956.4499318).
- [Wu77] F.F. Wu. “Theoretical study of the convergence of the fast decoupled load flow”. In: *IEEE Transactions on Power Apparatus and Systems* Volume 96 (1977), pp. 268–275. DOI: [10.1109/T-PAS.1977.32334](https://doi.org/10.1109/T-PAS.1977.32334).
- [ZC95] R.D. Zimmerman and Hsiao-Dong Chiang. “Fast decoupled power flow for unbalanced radial distribution systems”. In: *IEEE Transactions on Power Systems* Volume 10 (1995), pp. 2045–2052. DOI: [10.1109/59.476074](https://doi.org/10.1109/59.476074).
- [ZC97] Fan Zhang and C.S. Cheng. “A modified Newton method for radial distribution system power flow analysis”. In: *IEEE Transactions on Power Systems* Volume 12 (1997), pp. 389–397. DOI: [10.1109/59.575728](https://doi.org/10.1109/59.575728).