UC Riverside

UC Riverside Electronic Theses and Dissertations

Title

Analysis of Fluid Flows

Permalink

https://escholarship.org/uc/item/7qk5g6pn

Author

Luu, Huong

Publication Date

2024

Copyright Information

This work is made available under the terms of a Creative Commons Attribution-NonCommercial-NoDerivatives License, available at https://creativecommons.org/licenses/by-nc-nd/4.0/

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA
RIVERSIDE

Analysis of Fluid Flows

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Huong Luu

March 2024

Dissertation Committee:

Dr. Marek Chrobak, Chairperson
Dr. Ahmed Eldawy
Dr. Evangelos Papalexakis
Dr. Philip Brisk

Copyright by
Huong Luu
2024

The Dissertation of Huong Luu is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I would like to express my gratitude to my advisor, Dr. Marek Chrobak. His unwavering guidance and patience have been crucial in shaping the outcome of this research. Without his support, I would not have been here.

I would also like to thank the other members of my dissertation committee, namely Dr. Ahmed Eldawy, Dr. Evangelos Papalexakis, and Dr. Philip Brisk, for their valuable insights and thoughtful feedback.

Finally, I want to thank my family, fur babies, and friends who have provided encouragement and support throughout this academic journey. Your belief in me has been a constant source of inspiration and motivation.

ABSTRACT OF THE DISSERTATION

Analysis of Fluid Flows

by

Huong Luu

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, March 2024
Dr. Marek Chrobak, Chairperson

Fluid dynamics is the field of study that examines the motion of fluids such as liquids and gases. It can be used to investigate large-scale phenomena, such as ocean currents, as well as small-scale systems, like blood circulation. Fluid flows can be classified into two broad categories: laminar and turbulent flows. Laminar flows are smooth and streamlined, while turbulent flows are irregular and unpredictable. One of the fundamental tasks in analyzing fluid flows is to determine the flow rates and pressure values in a flow network, given its topology, channel dimensions, fluid properties, and boundary conditions.

In the first project, we study fluid mixing in microfluidic chips (MFCs), which are micro-scale fluid systems. In MFCs, flows are laminar, and for laminar flows, computing flow rates and pressure values are straightforward, but simulating the mixing process is computationally challenging. We present an approach for modeling concentration profiles in grid-based MFCs. Our algorithm outperforms COMSOL Multiphysics® software — commercial software that uses finite element analysis method to model physics processes — in terms of runtime while producing results that approximate those of COMSOL.

In the second project, we study turbulent flows in large-scale pipe systems such as water distribution systems and sewage networks. Unlike laminar flow systems, solving flows in turbulent models involves a system of nonlinear equations, and iterative algorithms have been widely applied in practice. We focus on the Hardy Cross loop-based algorithm (HC-loop) and the Newton-Raphson loop-based algorithm (NR-loop). We provide a mathematical analysis of the local convergence of these two algorithms, showing that, under certain conditions, NR-loop algorithm achieves quadratic convergence while HC-loop algorithm only converges linearly. This confirms earlier experimental observations reported in the literature.

In the third project, we investigate the minimum spanning tree congestion problem (STC), motivated by its application to improve the efficiency of the NR-loop algorithm for pipe flows analysis. We study the complexity of K-STC (STC for a fixed integer K) and prove that K-STC is $\mathbb{NP}$-complete for $K \geq 5$, improving the earlier hardness result and leaving only the case $K = 4$ open. We also investigate K-STC restricted to graphs of radius 2, establishing that this variant is $\mathbb{NP}$-complete for $K \geq 6$. Additionally, we explore a variant of STC, denoted K-STCD, where the objective is to determine if a graph has a depth-D spanning three of congestion K. We provide a tight bound for bipartite graphs by proving that 6-STC2 is $\mathbb{NP}$-complete, while 5-STC2 is solvable in polynomial time. Finally, we present polynomial-time algorithms for two special cases involving bipartite graphs with restrictions on vertex degrees.

Contents

List of Figures

Chapter 1

Introduction

Fluid dynamics is a fundamental discipline of physics and engineering that studies the motion, interactions, and phenomena observed in fluids, including liquids and gases. Its applications span various fields, such as medicine, bioengineering, civil engineering, meteorology, and many others. Fluid motion is broadly categorized into two types: laminar flows and turbulent flows. Laminar flows exhibit a smooth, ordered pattern, whereas turbulent flows are characterized by unpredictable and chaotic movement. Laminar flows occur when the flows are slow, the fluid is viscous, and the channels are small, as opposed to turbulent flows, which occur when the flows are fast, the fluid has low viscosity, and the channels are large. In this work, we study the underlying physical principles, mathematical foundations, and practical applications associated with the two distinct flow regimes.

One of the objectives of the analysis of fluid dynamics is to compute the flow rates and the pressure values within a fluid system. This task can be achieved by solving a system of governing equations. The two flow regimes share two physical laws: the flow conservation

and the energy conservation. The flow conservation states that at each node the total inflow is equal to the total outflow. The energy conservation dictates that around a closed loop, the pressure difference is zero. The main difference between the two flow models lies in the equation that relates the pressure loss between two ends of a channel to the flow rate in that channel. In the laminar flow regime, this relationship is linear, as opposed to being quadratic in the turbulent flow regime.

In laminar flow models, computing the flows and pressure values is straightforward while simulating the mixing process is more challenging. We study fluid flows and fluid mixing in microfluidic chips, which fall under the category of laminar flow models. Specifically, we present an algorithm to model fluid mixing in grid-based microfluidic chips, a topic extensively discussed in Chapter 2[1].

For turbulent flow models, the flow analysis problem involves a system of nonlinear equations. There are different formulations of this system, and we focus on the loop method. The prevalent choice to solve the loop equations is by using iterative algorithms. Two of the most widely used iterative algorithms for flow analysis problems are the Hardy Cross loop-based algorithm (HC-loop) and the Newton-Raphson loop-based algorithm (NR-loop). We present local convergence analysis of these two algorithms in Chapter 3.

One way to speed up the NR-loop algorithm is to use a sparse edge-cycle matrix. Motivated by sparsifying this matrix, we study the spanning tree congestion problem (STC) since a small congestion spanning tree corresponds to a cycle basis with a sparse edge-cycle matrix. We present several complexity results on STC in Chapter 4[2].

[1] A portion of this work appeared in [37]
[2] A portion of this work appeared in [38]

In the following sections of this chapter, we provide definitions, terminology, and literature reviews for the three problems mentioned above.

1.1 Microfluidics

Microfluidics is an emerging technology for manipulating nanoliter-scale fluid volumes, with applications in a variety of fields including biology, chemistry, biomedicine, and materials science. Fast progress in this area led to the development of microfluidic chips (MFCs), which are integrated microfluidic devices that are increasingly often used in various laboratory processes such as medical diagnosis [59], DNA purification [30], or cell lysis [29]. MFCs offer a solution to automate laboratory experiments, saving time, reducing labor costs, limiting the usage of chemical reagents, and replacing complex and expensive equipment [54, 29].

The technologies that are currently used in MFCs fall into two broad categories: droplet-based chips, where the fluid is manipulated in discrete units called droplets, and flow-based chips, based on continuous flow. We will focus exclusively on the flow-based model.

1.1.1 Continuous-flow MFCs

Fluid flow within enclosed micro-channels in continuous-flow MFCs is typically constant and continuous, usually driven by external pumps integrated into inlets and outlets of the chips. In this model, fluid is pumped into the chip's inlets at a constant velocity. The fluid is in a steady state, which is the condition where the flows at any point in the

system do not change over time. In such small-scale channels, the Reynolds number, which represents the ratio between inertial and viscous forces and predicts flow patterns, becomes very low, indicating laminar flow. In the laminar flow regime, fluid particles move in smooth layers with little interaction with adjacent layers. A fundamental physical law governing laminar flow is the Hagen–Poiseuille equation: $\Delta_e p = \mu_e q_e$ where $\Delta_e p$ is the pressure drop between two endpoints of a channel e, μ_e is the flow resistance, and q_e is the volumetric flow on e.

1.1.2 Fluid mixing

One of the key functions often implemented on MFCs is fluid mixing. Fluid mixing is particularly important in sample preparation, where the objective is to dilute the sample fluid, also called *reactant*, using another fluid that we refer to as *buffer*. For example, in cell lysis, the sample preparation process includes a step of mixing blood samples with citrate buffer [29]. For some experimental processes, samples with multiple pre-specified volumes and concentrations are needed. For instance, such a sample may consist of 5µL of reactant with concentration 10%, 10µL of reactant with concentration 20%, and 10µL of reactant with concentration 40%. Samples involving such multiple target concentrations are common in preclinical drug development processes, and they are also used for other experiments, for example, in biochemical assays [8]. For these applications, one needs to design an MFC that produces the specified target set of concentration/volume pairs of reactant.

1.1.3 Literature review

Several flow-based designs have been proposed in the literature. In [45], the authors proposed an MFC that uses two-way valves to produce serial dilution. An electrokinetically driven MFC design was introduced in [26] for serial mixing. The above approaches require different designs by changing valves, splitter channels placement, or tuning voltage control to create different target sets. In [9], the authors gave a dilution algorithm for given target concentration ratios using rotary mixers. However, their method produces waste, and it also uses valves which can complicate the fabrication process.

Grid mixers

A very different approach was developed by Wang *et al.* [57]. Their proposed solution involves creating a library of ready-to-use micromixers that users can query to find chip designs with desired properties. Their MFCs are simple rectilinear grids with two inputs (one for reactant and one for buffer) and three outputs, thus capable of producing a set of three different concentrations. They do not require any valves nor any other functional elements. A user identifies an appropriate design by submitting a query consisting of the desired reactant concentrations. In their approach, the design process is eliminated, and the database is created by exploring a large collection of randomly generated grids. For each random grid, its outlet concentration values are computed by simulating fluid flow through the grid using COMSOL Multiphysics® software. As shown in [57], these COMSOL simulation results provide very accurate predictions of outlet concentrations in actual fabricated MFCs.

Exploring such random designs is extremely time-consuming, as most randomly generated designs are actually not useful, either because they are redundant or because they produce concentrations that are of little interest (say, only near-pure reactant or buffer). Thus, to produce the desired number of designs for the grid library, one may need to examine many orders of magnitude more random designs. Indeed, in our experiment, to produce a collection of 2600 sufficiently different concentration triplets, we needed to generate 50 million 12×12 random grid designs.

With the need to test so many designs, this approach can only be used for small size grids because of the simulation bottleneck of COMSOL. The simulation time for each design in [57] is roughly a minute. It is also not scalable to bigger grids. COMSOL takes about 6 minutes to run a simulation for a 12×12 grid with the same mesh setting. The process can be sped up by using a coarser mesh, but this results in degraded accuracy of concentration predictions at the outlets. Further, some users may prefer to design custom grids for their choices of the attributes: velocity, solute, outlets' locations, diffusion coefficient of reactant, etc. Such users would need to have access to an often costly computational fluid dynamics software in order to run the simulations.

The approach based on random grid generation was also considered in [28], where the authors propose a method to remove redundant channels to make the design process more efficient, simplifying the fabrication of grid MFCs and reducing reactant usage.

1.1.4 Overview of contributions

Addressing this performance bottleneck in populating the grid library in the approach from [57], we developed an algorithm to model fluid mixing in microfluidic grids in

order to predict the reactant concentrations at the outlets. In our approach, concentration profiles in grid channels are approximated using a simple 3-piece linear function, which allows us to simulate the mixing process in time linear in the grid size. The overall algorithm is scalable, simple, and produces good approximation of concentration values and flow rates at outlets in grid-based microfluidic chips, as compared to the results from COMSOL Multiphysics®. It is also much more general than the model from [57, 28], as it allows grids of all sizes, any number of outlets, arbitrary inflow velocities, and arbitrary fluids.

1.2 Pipe networks analysis

The analysis of pipe networks is a well-studied problem in civil engineering that arises in the design and modeling of large-scale hydraulic networks, such as municipal water systems, sewage systems, or substance distribution networks. In the most common formulation, the objective is to determine the flow rates and pressure differences in all pipe segments within the network, given its topology, nodal inflows/outflows, and the parameters of the pipes and fluid (diameter, friction, fluid density, etc.) [55]. At a fundamental level, this problem is analogous to the analysis of electrical flow in circuits and fluid flows in MFCs. These problems share similar physical principles: mass (or flow) conservation and energy conservation. However, in contrast to their electrical and MFC counterparts, in large-scale pipe networks, the equation governing the relationship between flow and pressure difference in a pipe segment is nonlinear. This equation, known as the Darcy-Weisbach equation, asserts that the pressure difference in a pipe segment e is proportional to the square of its flow: $|\Delta_e p| = \mu_e q_e^2$, where $|\Delta_e p|$ is the positive pressure difference, q_e is the flow value,

and μ_e is a constant coefficient that is inversely proportional to the pipe's diameter, and proportional to the friction factor, pipe length, and fluid density.

Pipe network analysis boils down to solving the system of the above-mentioned equations, namely the flow conservation equations for all nodes and the energy conservation equations for all pipe segments. The specific choice of variables, be it pressure values or flow values, leads to different formulations of these equations, with three common approaches known as the loop method, the flow method, and the node method [12]. (See Section 3.1 and Appendix 3.5.) We focus mainly on algorithms based on the loop method. This method assumes that some initial flow satisfying the flow conservation conditions is given, and its unknowns are the flow adjustments along the network's cycles with respect to this initial flow. (See Section 3.1 for a detailed description). Note that in either formulation, the resulting system of equations is nonlinear, and computing its exact solution is not feasible. Instead, iterative techniques that compute approximate solutions have been widely adopted in the scientific literature and in practice.

1.2.1 Literature review

One of the earliest approaches to pipe network analysis was proposed by Cross in 1936 [16]. He introduced both the loop and node variants of his method, which he referred to as the methods of balancing flows and balancing heads. We denote them as *HC-loop* and *HC-node*, respectively. Due to its simplicity, Algorithm HC-loop has been used to analyze small networks with spreadsheets or hand calculations. Another frequently used approach involves applying the generic Newton-Raphson algorithm for solving nonlinear equations.

This algorithm can be applied to any of the above three settings: flow, loop, or node, and we use notations *NR-flow*, *NR-loop* and *NR-node* for their respective variants.

While there is a fair amount of literature on the experimental performance of these algorithms, relatively little is known about their provable convergence properties. In [16], Cross claimed that Algorithm HC-loop's convergence is "sufficiently rapid," though this claim was not supported by rigorous analysis. Adams [1] provided an analysis of the accuracy of the flow corrections generated by Algorithm HC-loop by comparing it with the exact correction values. This result is useful for the convergence analysis of simple systems that consist of only one cycle; however, for larger systems, the mathematical analysis is incomplete. Empirical results in [23] show fast convergence of HC-loop for several networks in the dataset. Altman and Boulos report in [2] an attempt to analyze the local convergence of NR-flow; however, as we explain in Section 3.4, their analysis is invalid. In [13], Brkic conducted a comparative study of NR-loop and HC-loop, and his experimental results support the earlier observations that NR-loop exhibits faster convergence.

As for other methods, Shamir and Howard commented in [49, 50] that Algorithm NR-node might not converge if it encounters a singular derivative matrix or if it enters an infinite loop, although no specific examples were given there. In Section 3.5, we provide an example of which Algorithm NR-node enters an infinite loop. A fairly extensive empirical comparison of different formulations of both Algorithm HC and Algorithm NR was presented by Wood [58]. The results indicate that the loop methods encounter fewer convergence issues than the respective node methods and that the Newton-Raphson algorithm is more reliable for all formulation variants.

1.2.2 Overview of contributions

We present an analysis of the local convergence of Algorithms NR-loop and HC-loop in Section 3.2 and Section 3.3. We show that Algorithm NR-loop converges quadratically fast if the initial flow is sufficiently close to the solution, and we provide a bound on the radius of its quadratic convergence. The quadratic convergence of the Newton-Raphson method is not surprising — in fact, it is known to converge quadratically under some fairly mild assumptions. Our aim here was to correct and refine the analysis of the convergence radius provided in paper [2].

For Algorithm HC-loop, we prove a similar local convergence property, although in this case, the convergence rate is only linear. We also show that this bound is tight by exhibiting an example where Algorithm HC-loop's convergence is not better than linear. These results provide a theoretical confirmation of experimental observations discussed earlier about the superiority of NR-loop over HC-loop.

1.3 The minimum spanning tree congestion problem

Problems involving constructing a spanning tree that satisfies certain requirements are among the most fundamental tasks in graph theory and algorithmics. One such problem is the *spanning tree congestion problem*, STC for short, which has been studied extensively for many years. In this problem, we seek a spanning tree T of a given graph G that roughly approximates the connectivity structure of G, in the following sense: Embed G into T by replacing each edge (u, v) of G by the unique u-to-v path in T. Define the *congestion of an*

edge e of T as the number of such paths that traverse e. The objective of STC is to find a spanning tree T in which the maximum edge congestion is minimized.

1.3.1 Literature review

The general concept of edge congestion was first introduced in 1986, under the name of *load factor*, as a measure of the quality of an embedding of one graph into another [7] (see also the survey in [48]). The problem of computing trees with low congestion was studied by Khuller *et al.* [32] in the context of solving commodities network routing problems. The trees considered there were not required to be spanning subtrees, but the variant involving spanning trees was also mentioned. In 2003, Ostrovskii provided independently a formal definition of STC and established some fundamental properties of spanning trees with low congestion [42]. Subsequent to its introduction, numerous combinatorial and algorithmic findings of the problem have been reported in the literature. Otachi's comprehensive survey paper [43] offers an extensive and current overview of these developments.

As established by Löwenstein [36], STC is $\mathbb{NP}$-hard. As usual, this is proved by showing $\mathbb{NP}$-completeness of its decision version, where we are given a graph G and an integer K, and we need to determine if G has a spanning tree with congestion at most K. Otachi *et al.* [44] strengthened this by proving that the problem remains $\mathbb{NP}$-hard even for planar graphs. In [40], STC is proven to be $\mathbb{NP}$-hard for chain graphs and split graphs. On the other hand, computing optimal solutions for STC can be achieved in polynomial time for some special classes of graphs: complete k-partite graphs, two-dimensional tori [35], outerplanar graphs [11], and two-dimensional Hamming graphs [34].

We focus our study on the decision version of STC where the bound K on congestion is a fixed constant. We denote this variant by K-STC. Several results on the complexity of K-STC were reported in [44]. For example, the authors of [44] show that K-STC is decidable in linear time for planar graphs, graphs of bounded treewidth, graphs of bounded degree, and for all graphs when $K = 1, 2, 3$. On the other hand, they show that the problem is $\mathbb{NP}$-complete for any fixed $K \geq 10$. In [10], Bodlaender *et al.* proved that K-STC is linear-time solvable for graphs in apex-minor-free families and chordal graphs. They also show an improved hardness result of K-STC, namely that it is $\mathbb{NP}$-complete for $K \geq 8$, even in the special case of apex graphs that only have one unbounded degree vertex. As stated in [43], the complexity status of K-STC for $K \in \{4, 5, 6, 7\}$ remains an open problem.

Very little is known about the approximability of STC. The trivial upper bound for the approximation ratio is $n/2$ — this ratio is achieved, in fact, by any spanning tree [43]. As a direct consequence of the $\mathbb{NP}$-completeness of 8-STC, there is no polynomial-time algorithm to approximate the optimum spanning tree congestion with a ratio better than 1.125 (unless $\mathbb{P} = \mathbb{NP}$).

Other related work The spanning tree congestion problem is closely related to the tree spanner problem, in which the objective is to find a spanning tree T of G that minimizes the stretch factor, defined as the maximum ratio, over all vertex pairs, between the length of the path in T and the length of the shortest path in G connecting these vertices. In fact, for any planar graph, its spanning tree congestion is equal to its dual's minimum stretch factor plus one [20, 44]. This direction of research has been extensively explored, see [14, 18, 19].

As an aside, we remark that the complexity of the tree 3-spanner problem has been open since its first introduction in 1995 [14].

STC is also intimately related to problems involving cycle bases in graphs. As each spanning tree induces a fundamental cycle basis of the given graph, a spanning tree with low congestion yields a cycle basis for which the edge-cycle incidence matrix is sparse. The sparsity of such matrices is desirable in linear-algebraic approaches to solving some graph optimization problems, for example, analyses of distribution networks such as pipe flow systems [3].

STC can be thought of as an extreme case of the graph sparsification problem, where, given a graph G, the objective is to compute a sparse graph H that captures connectivity properties of G. Such H can be used instead of G for the purpose of various analyses, to improve efficiency. See [6, 21, 53] (and the references therein) for some approaches to graph sparsification.

1.3.2 Overview of contributions

In Section 4.2, we provide an improved hardness result for K-STC. Theorem 11 combined with the results in [44] leaves only the status of 4-STC open. This theorem also implies a better lower bound on the approximation ratio for STC, which will be discussed in Section 4.2.

A common feature of the hardness proofs for STC, including ours, is that they all use graphs of small constant radius (or, equivalently, diameter). Another property of STC that makes its approximation challenging is that the minimum congestion value is not monotone with respect to adding edges. The example graph in [42] showing this non-

monotonicity is also of small radius (in fact, only 2). These observations indicate that a key to further progress may be in better understanding of STC in small-radius graphs.

This motivates our additional hardness result presented in Section 4.3, where we focus on graphs of radius 2. (For radius 1, the problem is trivial.) We prove that K-STC remains $\mathbb{NP}$-complete for this class of graphs for any fixed integer $K \geq 6$. In fact, this holds even if we further restrict such graphs to be bipartite and have only one vertex of non-constant degree.

Probing further in this direction, in Section 4.4, we consider the variant of STC denoted K-STCD, in which the objective is to determine if the graph has a spanning tree of depth at most D and congestion at most K. Note that this is not a restriction of STC, as the minimum congestion for trees of depth 2 can be larger than the optimum value of STC. We observe that our $\mathbb{NP}$-completeness proof in Section 4.3 can be adapted to prove that K-STC2 is $\mathbb{NP}$-complete for $K \geq 6$. This is true even if input graphs are restricted to bipartite graphs with only one vertex of non-constant degree. For bipartite graphs, we establish a tight bound by proving that 5-STC2 is polynomial-time solvable. Complementing this, we present two other natural special cases involving bipartite graphs and restrictions on vertex degrees, in which the optimal congestion spanning tree can be computed in polynomial time in Section 4.5.

Chapter 2

Modeling fluid mixing in microfluidic grids

In this chapter, we present an approach to model fluid mixing in grid-based microfluidic chips (MFCs), an idea developed by Wang *et al.* [57]. Our algorithm efficiently predicts the flow velocities and concentrations at the outlets of the MFCs with good accuracy compared to COMSOL Multiphysics® software.

2.1 Statement of the problem

We study grid-based MFCs for fluid mixing introduced in [57]. Their model is 8×8 grids with 2 inlets along the top edge of the grid and 3 outlets at the bottom, as shown in Figure 2.1(a). The left inlet contains a reactant, with concentration value 1, and the right inlet contains a buffer, with concentration value 0. The channel width is 0.2 mm, and the channel length (distance between two grid vertices) is 1.5 mm. The fluid velocity in the inlets is 10 mm/s. The outlets' pressure is 0 Pa. The reactant is either sodium, fluorescein, or bovine serum albumin.

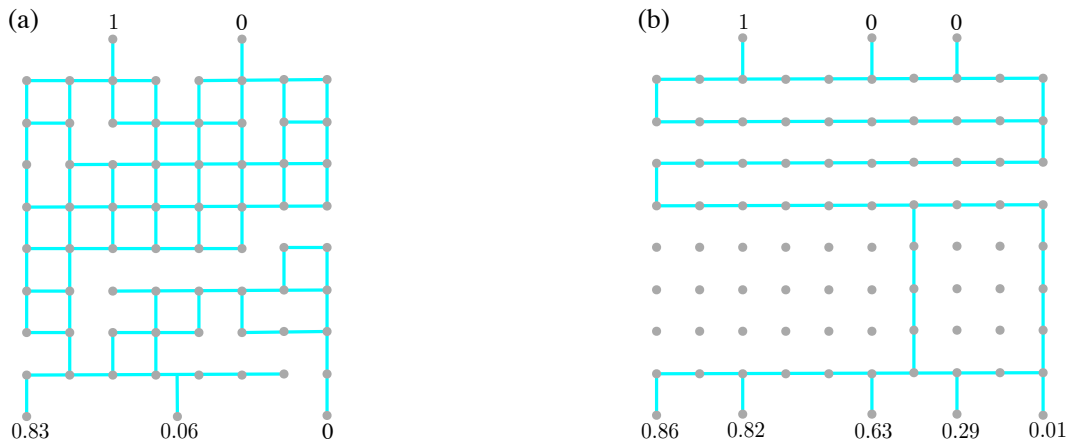


Figure 2.1: Examples of (a) an 8x8 grid from [57], and (b) a 10x8 grid in our model

We generalize the model from [57] in several ways. We allow arbitrary $m \times n$ grids with any number of inlets and outlets (see Figure 2.1b). The inlets are located along the top edge of the grid, and the outlets are at the bottom. The inlet solutions can take any concentration from 0 to 1, but they must satisfy the following *inlet monotonicity property*: the inlet concentrations need to be non-increasing from left to right. The inflows are of a given constant rate, and the pressure values at all outlets are 0. Our model assumes that the flow throughout the grid is laminar, which is the case in standard microfluidic applications.

In this setting, the problem we address can be formulated as follows: We are given (1) the specification of a grid design, and (2) fluid properties, namely its concentration and velocity at each inlet and its diffusion coefficient. The goal is to determine the fluid concentration and velocity values at the outlets.

2.2 Overview of the algorithm

Our algorithm for predicting reactant concentrations at the outlets is based on modeling its concentration profiles in the grid's channels. Such a concentration profile is a function that represents concentration values of the reactant along a line perpendicular to the channel. When fluid flows through straight segments of the grid, this profile changes according to the laws of diffusion. In a node of the grid, a flow may be split, or several flows may be joined, and the profile changes accordingly, producing complex nonlinear functions. The main idea behind our algorithm is to approximate this profile using a simple 3-piece linear function. Once the profile at an output channel is computed, it determines the reactant concentration at this outlet.

We now give an overview of our algorithm, with more detailed descriptions given in the sections that follow.

1. Verify the correctness of the grid design, namely whether each edge (channel) is on at least one inlet-to-outlet path. (In our implementation, spurious fragments of the grid are automatically removed.)

2. Compute the flow rates in each channel and pressure values at each node (see Section 2.4).

3. Partition the grid into parts, each part being either a straight channel or a node. Depending on flow direction, a node can be one of three types: a join node (2-way or 3-way), a split node (2-way or 3-way), or a combined join/split node (with 2 inflows and 2 outflows). Sort these parts in an order consistent with the flow direction. (Once

the flows are computed, one can think of the grid design as an acyclic directed graph. The desired order is then any topological sort of this acyclic graph.)

4. Process the grid parts, in the earlier determined order, computing approximate concentration profiles (see Section 2.5):

   - For straight channels, the concentration profile at the end of the channel is determined from the profile at the beginning of the channel based on the time the flow spends in this channel. This time is computed from the channel length and flow velocity.
   - For nodes representing flow splits, split the incoming profile into outgoing profiles according to flow velocity ratios.
   - For nodes representing flow joins, join the incoming profiles into the outgoing profile according to velocity ratios. This outgoing profile is then approximated by a 3-piece linear function.

5. Once all flow profiles in the grid are determined, for each outlet, compute its fluid concentration as the integral of its concentration profile divided by the channel width.

Running time The algorithm for profile computations takes only constant time to update the profile for each node and channel, so the running time of this step is linear with respect to the size of the grid design. (Thus, it is never worse than $O(mn)$ for an $m \times n$ grid.) The overall running time is dominated by solving the linear system in part (2). For grid sizes that might be of use in grid libraries, say up to 20×20, Gaussian elimination is sufficiently

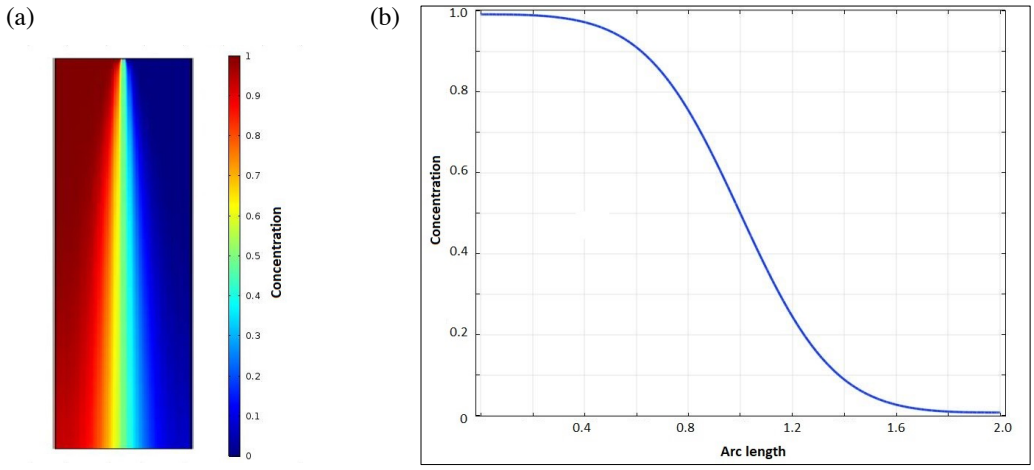


Figure 2.2: (a) The result of mixing of reactant (red) and buffer (blue) due to diffusion. (b) The graph representing the concentration profile of this mixture at the end of the channel.

fast. For larger grids, one can take advantage of the sparsity of the linear system to speed up the computation.

2.3 Concentration profile model

2.3.1 True concentration profiles

Consider a mixture of two fluids, one reactant (with concentration 1) and the other buffer (with concentration 0), flowing along a straight channel of some width w and length l. For any distance $l' \leq l$ from the beginning of this channel, a *concentration* profile at l' is a function that gives concentrations of all points in the channel along the line segment (of length w) that is perpendicular to the channel and directed counter-clockwise to the flow. We will be interested in how this profile evolves with time, that is with the value of l' increasing.

Figure 2.2 shows an example. Reactant and buffer are injected at the same rate into the top opening of a vertical straight channel and allowed to mix while they flow. Initially, the two fluids are separated, with the reactant to the left of the buffer, so the profile will be a 1/0 function. The flow is assumed to be laminar, and the two fluids will gradually mix as a result of diffusion. After a period of time, this mixing produces a non-uniform concentration profile as shown in Figure 2.2(b). This concentration profile is a smooth curve with the leftmost region having concentration 1, the rightmost region having concentration 0, and the middle region contains partially mixed fluids with concentration decreasing from left to right. Using the diffusion model, this profile function can be determined from the mixing time, which is the time it takes for the fluid to flow through the channel. Half of the width of this middle region is referred to as diffusion length and denoted L (normal to the flow direction, units m). It can be computed from the formula (see [33]):

$$L = 2\sqrt{Dt}. \tag{2.1}$$

where t is the mixing time and D is the diffusion coefficient of the fluid (units m^2/s).

In microfluidic grids, the flow may be repeatedly split, or different flows may get combined, and the resulting concentration profile will not have the form in Figure 2.2(b) anymore; in fact, profile functions that arise in such grids are too complex to be captured analytically. Below, we prove, however, that these profiles have a certain monotonicity property that will allow us to approximate them by a simpler function. Interestingly, this monotonicity property involves the concept of the partial order that is dual to the flow pattern.

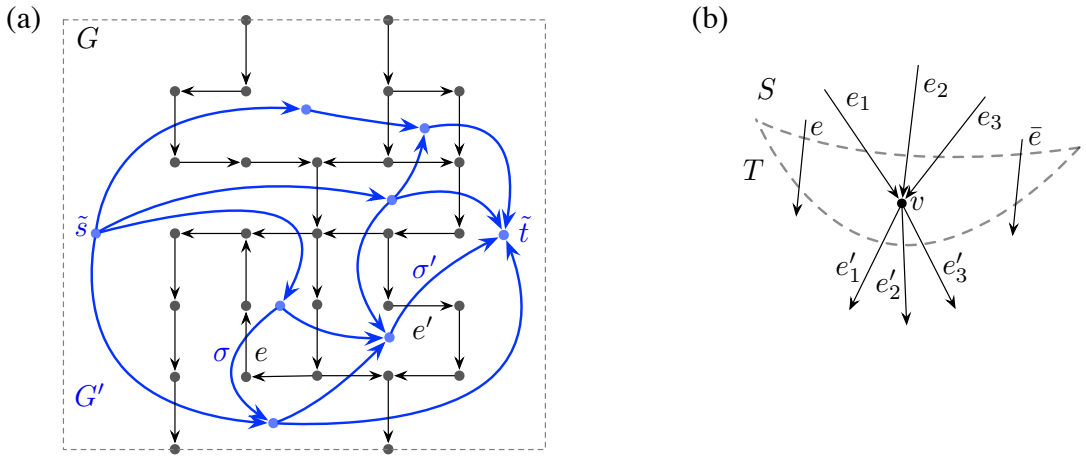


Figure 2.3: (a) Grid representation graph G (black) and its dual graph $\tilde{G}$ (blue). (b) Illustration of moving a vertex v from T to S.

2.3.2 Concentration monotonicity

The intuition behind the monotonicity property is illustrated in Figure 2.2(b); intuitively, the profile function in a channel should be monotonely decreasing from left to right. This property is trivial if we start with a 1/0 profile (with pure reactant to the left of pure buffer) at the top of a vertical channel and allow the fluid to diffuse when it flows down along the channel. However, in a mixing grid, the flow pattern may be quite complex. For example, depending on a grid's structure and flow velocities at its inlets, the flow direction in a vertical channel could be down or up. As a result, even the notions of "left" and "right" are not well defined anymore. Joins and splits complicate this issue even more. Thus, to define this monotonicity property, we need to capture the notion of "left-to-right direction" not only with respect to one channel, but also between different channels. This notion will be formalized using a partial ordering of the grid's channels. This partial order is defined as the dual order of the flow pattern in the grid. Below, we formalize these concepts.

Once the flow directions are computed, the flow pattern through the grid design can be naturally represented as a straight-line planar drawing of a DAG (directed acyclic graph) whose nodes are grid points (including inlets and outlets) and edges are channels with directions determined by the flow. We will denote this graph by G.

Next, we construct a dual DAG $\tilde{G}$. To this end, enclose the grid in an axis-parallel rectangle, slightly wider than the grid, with the inlets on its top edge and outlets on its bottom edge, as in Figure 2.3(a). The grid (that is, the embedding of G) partitions this rectangle into regions. For each region of G, we create a vertex of $\tilde{G}$. Two vertices ϕ, ψ of $\tilde{G}$ are connected by an edge if the boundaries of their corresponding regions share at least one edge of G. The direction of the edge between ϕ and ψ is determined as follows: pick any edge (u, v) of G shared by the regions of ϕ and ψ. This edge (u, v) must have a different orientation in the boundaries of ϕ and ψ (clockwise in one and counter-clockwise in the other). Then the edge between ϕ and ψ is directed from the node where (u, v) is clockwise to the node where it is counter-clockwise. This definition does not depend on the choice of (u, v). We will refer to this edge as being dual to (u, v). It can be shown that $\tilde{G}$ is a DAG with a unique source $\tilde{s}$ and unique sink $\tilde{t}$ that correspond to the regions on the left and right of the grid design, respectively. (Except for secondary technical differences, the construction of such a dual can be found, for example, in [56, 5].)

We now use $\tilde{G}$ to define a partial order on the edges of G (that is, the channels of our grid design). Call two edges e, e' of G *related in* G if they are both on the same inlet-to-outlet path; otherwise, call them *unrelated in* G. If e and e' are unrelated in G then, denoting by σ and σ' their dual edges, there must exist a path from $\tilde{s}$ to $\tilde{t}$ in $\tilde{G}$ that

contains σ and σ'. If σ is before σ' on this path, then we write $e \trianglelefteq e'$ and say that e *dually precedes* e'. Figure 2.3(a) shows an example. It is not difficult to verify that relation "$\trianglelefteq$" is a partial order on G's edges.

Theorem 1. *(Concentration profile monotonicity.) Consider a grid design as described in Section 2.1 (in particular, the inlet concentrations are non-increasing) with some flow, and its corresponding graph G. The concentration profiles in G satisfy the following properties:*

(cpm1) For any edge e of G, each concentration profile across e is a non-increasing function.

(cpm2) For any two edges e, e' of G, if $e \trianglelefteq e'$, then each concentration value in each profile across e is at least as large as each concentration value in each profile across e'.

Proof. The formal proof of Theorem 1 is omitted here due to lack of space; we only give a brief sketch. For any two edges e, e' of G, we write $e \succcurlyeq e'$ to indicate that all concentration values in each profile across e are as large as all concentration values in each profile across e'. With this definition, part (cpm2) says that $e \trianglelefteq e'$ implies $e \succcurlyeq e'$.

Observe first that property (cpm1) is preserved as we move a profile along e in the flow direction due to the properties of mixing (which shifts reactant mass from higher to lower values). With this in mind, it is sufficient to arbitrarily select one "representative" profile for each edge and to prove properties (cpm1) and (cpm2) only for these selected profiles.

Define a cut (S, T) of G in a natural way, as the partition of its vertices such that each inlet-to-outlet path visits vertices in S before visiting vertices in T. We prove by induction on $|S|$ that all edges with tail vertex in S satisfy the theorem. This is true

in the base case when S contains only inlets. In the inductive step, the definition of cuts implies that there is a vertex v that has all predecessors in S. Choose any such v. We move v from T to S (see Figure 2.3b) and show that the inductive claim is preserved. The clockwise ordering of v's incoming edges $e_1, e_2, ...$ is the same as their $\trianglelefteq$ order. The same applies to the counter-clockwise order of its outgoing edges, $e'_1 \trianglelefteq e'_2 \trianglelefteq ....$ We also know that the concentration profile at v is the joined profile of $e_1, e_2, ...$ and then it is split into decreasing concentration profiles of $e'_1, e'_2, ...$ If there exist edges e and $\bar{e}$ in S that satisfy $e \trianglelefteq e_1 \trianglelefteq e_2 \trianglelefteq ... \trianglelefteq \bar{e}$, then e and $\bar{e}$ are also a predecessor and successor, respectively, of $e'_1, e'_2, ...$ in the $\trianglelefteq$ ordering. These observations and the inductive assumption applied to e and $\bar{e}$ imply that $e \succcurlyeq e'_1 \succcurlyeq e'_2 \succcurlyeq ... \succcurlyeq \bar{e}$. □

2.3.3 Approximate concentration profiles

In our technique, we approximate concentration profiles by simple 3-piece linear functions specified by four parameters: a_L, a_R, d_L, and d_R, as shown in Figure 2.4(a). In interval $[0, d_L]$, the concentration is a_L; in interval $[w-d_R, w]$, the concentration is a_R, and the concentration linearly decreases in interval $[d_L, w-d_R]$, from a_L to a_R. Throughout the paper we will refer to such simplified profile functions as *SP-functions*. We have $w - d_L - d_R = 2L$, where L is the diffusion length value. The area under the profile curve, divided by the channel width w, represents the overall (average) concentration value of the fluid in this channel.

Figure 2.4: (a) An SP-function profile. (b) Grid partition: a join node on the top, followed by a downward straight channel and then a split node.

We remark that Theorem 1 remains valid for our approximate concentration profiles instead of real ones. This is because its proof remains correct for any (not necessarily physically valid) mixing process that preserves the mass of reactant and buffer, and moves reactant mass rightward over time. Our approximate profile model has this property.

In Section 2.5, we explain how we can use SP-functions to compute approximate concentration profiles for each part of the grid design.

2.4 Computing flows

The computation of the flow direction, flow rate, and pressure value in each channel of the grid is quite straightforward, and it can be achieved by solving a system of linear equations. The unknowns are pressure values at every grid node and flow velocities in every channel. The first set of equations in this system are flow conservation equations: at every grid node the total inflow is equal to the total outflow. The second set of equations are Hagen-Poiseuille equations which give the relationship between the flow rate and the pressure drop between the two ends of a channel e: $|\Delta_e p| = \mu_e q_e^2$, where q_e is the volumetric

flow, and R is the flow resistance. As the resistance value is the same in every channel segment and the input data specifies the velocity at the grid inlets, the exact value for μ is not needed and can be assumed to be 1.

One thing to note is that in our setting, we assume there is no friction at the walls of the channel, thus implying that the velocity is uniform across the channel, which simplifies the formulas for updating the profiles. (With friction, the velocity profile is a parabolic function.) We found, however, that this assumption has only a negligible effect on the computed concentration values.

2.5 Approximate concentration profiles

The core of our algorithm is a procedure for updating approximate concentration profiles (represented by SP-functions) along the grid, namely part 4 of the overall algorithm in Section 2.2. We describe this procedure in this section.

As mentioned earlier in Section 2.2, the grid is partitioned into parts: straight channels and nodes, where nodes can be of several types depending on the flow directions of its channels: join nodes (2-way or 3-way), split nodes (2-way or 3-way), and combined join/split nodes (2 inflows and 2 outflows). This partitioning is illustrated in Figure 2.4(b). In this figure, the channels are cut at points p_1, p_2, p_3, p'_3, p_4 and p_5, producing one join node, one straight channel, and one split node.

In the join node at the top, flows 1 and 2 are joined into flow 3. The concentration profile at point p_3 is computed by combining SP-functions at points p_1 and p_2, taking the velocity ratios into account. The combined function may not be an SP-function, and if so, we approximate it by an SP-function.

The straight channel stretches from the beginning of the channel at point p_3 to its end point p'_3. The approximate SP-function at point p'_3 is computed from the SP-function at point p_3 using the diffusion formula. (To account for mixing in the grid nodes, instead of the original channel length l, the algorithm uses a slightly larger value $\tilde{l} = l + w/2$, with w is the width of the channel.)

In the split node at the bottom, we use the SP-function at point p'_3 and the flow velocities to compute the split SP-functions at points p_4 and p_5. This case is relatively simple, as splitting an SP-function profile produces SP-functions, so no additional simplification is needed here.

2.5.1 Straight channels

For a straight channel, the concentration profile changes over time with diffusion length increasing, that is with the middle portion of the SP-function gradually becoming longer and flatter. The SP-function may also change its form, with d_L becoming 0 or d_R becoming 0. We divide the evolution of the SP-function along a straight channel into time intervals short enough so that in each such time interval the profile function has the same form.

We now demonstrate how the SP-function changes in a straight channel during one of these time intervals. The fundamental idea behind our approach is this: Suppose that

Figure 2.5: Updating the concentration profile in a straight channel in (a) case 1, and (b) case 2. The "before" profile is black and the "after" profile is blue.

$d_L > 0$ and $d_R > 0$ in Figure 2.5a (Case 1 below). We then think of the current profile as being a result of a mixing process in a straight channel (or for stationary fluid) that started from the original 1/0-valued profile and lasted for some unknown time t. This allows us to compute t from the diffusion length formula (2.1). Once we have t, we can compute the new SP-function at a time $t' = t + \Delta t$, where Δt is chosen to be the time until the next "event," which is the time when either the end of the channel is reached or when the profile form changes (that is, d_L or d_R becomes 0). This leads to a relatively simple solution for updating concentration profiles that satisfy $d_L > 0$ and $d_R > 0$, that is for SP-functions that have three non-empty segments. If $d_L = 0$ or $d_R = w$, or both, a slightly less direct approach is required (Cases 2, 3 and 4 below).

Case 1: $d_L > 0$ and $d_R > 0$ (see Figure 2.5a).

As explained above, using formula (2.1) we compute $t = L^2/4D$. Next, we compute the new diffusion length L' after time Δt:

$$L' = 2\sqrt{D(t + \Delta t)} = \sqrt{L^2 + 4D\Delta t}$$

This will give us the new SP-function at time t', defined by parameters:

$$d'_{L} = d_{L} - (L' - L) \text{ and } d'_{R} = d_{R} - (L' - L)$$

We choose Δt to be the time until either the end of the channel is reached, or until one of d_{L}, d_{R} becomes 0, whichever happens first.

Case 2: $d_{L} = 0$ and $d_{R} = 0$ (see Figure 2.5b).

The channel boundaries introduce some distortion into the diffusion process that is difficult to model. Our approach here is based on the observation (verified experimentally) that this distortion has negligible effect on the overall profile. Thus, we think about the current concentration profile as the linearly decreasing part of the profile in a "virtual" wider channel in which the concentration profile has the same form as in Case 1, with $L = w/2 = 2\sqrt{Dt}$. The goal is to estimate the change of a_{R} and a_{L} in time interval $[t, t']$.

For $\tau \in [t, t']$, define $a_{L}(\tau)$ and $a_{R}(\tau)$ to be the concentrations at the left and right wall at time τ, and $\alpha(\tau) = (a_{L}(\tau) - a_{R}(\tau))/2$. We want to estimate the derivative $d\alpha/d\tau$.

To this end, let dL and $d\alpha$ denote the changes of L and $\alpha(\cdot)$ in a time interval $[\tau, \tau + d\tau]$, where $d\tau > 0$ is very small. (In the calculations below, we will approximate some values assuming that $d\tau$ approaches 0.) We have:

$$dL = 2\sqrt{D(\tau + d\tau)} - 2\sqrt{D\tau} = 2\sqrt{D}\, d\tau/(\sqrt{\tau + d\tau} + \sqrt{\tau}) \approx d\tau\, \sqrt{D/\tau}$$

and by simple geometry,

$$d\alpha = -\alpha(\tau)\, dL/(L + dL)$$

Figure 2.6: Updating the concentration profile in a straight channel in (a) case 3, and (b) case 4.

Substituting, we get:

$$d\alpha \approx -d\tau\, \alpha(\tau)/2\tau$$

Thus, the derivative is:

$$d\alpha(\tau)/d\tau = -\alpha(\tau)/(2\tau)$$

Solving this differential equation and using the initial condition at t gives us:

$$\alpha(\tau) = \alpha(t)\sqrt{t/\tau} \text{ where } \alpha(t) = (a_L - a_R)/2$$

Thus, at time t' we have:

$$\alpha(t') = \alpha(t)\sqrt{t/t'}$$

From this, we can calculate the new concentration values at the channel walls:

$$a'_L = a_L - \alpha(t)(1 - \sqrt{t/t'}) \text{ and } a'_R = a_R + \alpha(t)(1 - \sqrt{t/t'})$$

Here, t' is taken to be the time when the end of the channel is reached.

Case 3: $d_L > 0$ and $d_R = 0$ (see Figure 2.6a).

The new concentration profile in this case is computed based on the assumption that the value of d_L will decrease according to formula (2.1). This gives us the new value d'_L for the new SP-function, namely

$$d'_L = d_L - (L' - L) \text{ where } L' = 2\sqrt{D(t + \Delta t)} = \sqrt{4D\Delta t + (w - d_L)^2}$$

Then we use molecular preservation property (that is the area under the concentration profile does not change) and straightforward calculation to obtain the new concentration at the right wall,

$$a'_R = a_L - L(a_L - a_R)/L'$$

Here, Δt is taken to be the time until either the end of the channel is reached or until d_L becomes 0, whichever happens first.

Case 4: $d_L = 0$ and $d_R > 0$ (see Figure 2.6b).

This case is symmetric to Case 3, and a similar calculation gives us the values of d'_R and a'_L.

2.5.2 Joining concentration profiles

To join flows, we first simply combine together the SP-functions of two or three joined channels, with the portion of the channel width that each joined flow occupies being proportional to the velocities of the inflows, as shown in Figure 2.7(a). We will refer to this profile as the *combined profile*. In general, the combined profile will not be an SP-

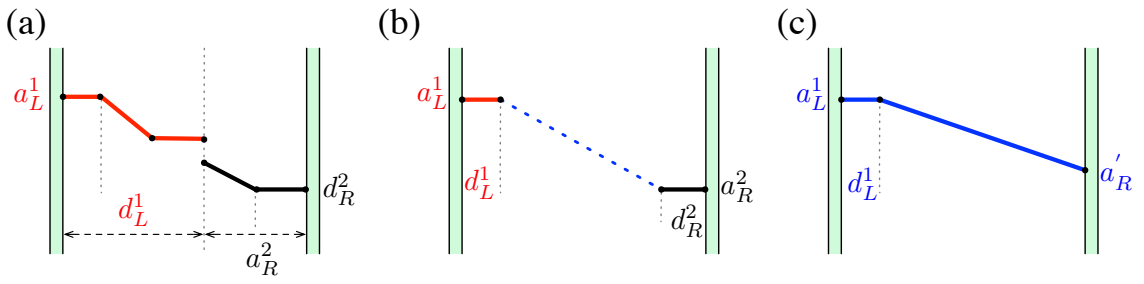


Figure 2.7: Example of joining concentration profiles of two flows (red and black). In (a), the combined profile, in (b) the tentative SP-function, and in (c) the final SP-function (in the case when the area under the tentative profile is too small).

function. If this is the case, we will need to simplify it to an SP-function. This will be done in two steps. First, we will convert the combined profile into a *tentative* SP-function (see Figure 2.7b), which will be later adjusted to satisfy the reactant volume preservation property.

The correctness of joining profiles depends critically on Theorem 1. This theorem implies that the concentration of the combined profile is non-increasing from left to right, as illustrated in Figure 2.7a. This figure shows two profiles being combined. The case of combining three profiles is essentially the same, as the middle channel's SP-function is only needed to compute the area under the combined function; its parameters can be ignored. Thus, to simplify the description below, we will assume that we are dealing with a 2-way join node.

Let a_L^1 and d_L^1 represent the parameters of the combined profile inherited from the corresponding parameters of the left joined channel. (So a_L^1 is the concentration along the left wall in the left channel, and d_L^1 is the length of its SP-function's left flat segment, but rescaled according to the channel's flow velocity.) By a_R^2 and d_R^2 we denote the correspond-

ing parameters inherited from the right joined channel. We take these four values as the parameters of our *tentative* SP-function profile (see Figure 2.7(b) for an example). This function is only tentative, because the area under this SP-function may be different than that under the combined profile.

The final SP-function, whose parameters will be denoted a'_L, d'_L, a'_R and d'_R, is computed from the tentative SP-function by adjusting its parameters to make sure that the reactant volume (the area under the profile) is preserved. This is done as follows. Let A be the area under the original combined profile (before converting it into the tentative form). If the area under the tentative profile is smaller than A, then the left segment of the final profile is the same as in the tentative profile, that is

$$a'_L = a^1_L \text{ and } d'_L = d^1_L$$

while the middle and right segments are adjusted to increase the area to A: We start with

$$d'_R = d^2_R \text{ and } a'_R = a^2_R$$

We then decrease d'_R until either the area equals A or d'_R is reduced to 0. If d'_R becomes 0, we then increase a'_R until the area becomes A. The case when the area under the tentative profile is larger than A is symmetric; in this case, the right segment of the final profile is the same as in the tentative profile, and we gradually lower the middle and left segment to reduce the area to A. See Figure 2.7 (c) for an example of a final SP-function.

Figure 2.8: (a) Splitting an SP-function into two SP-functions (red and black). (b) A join-and-split node.

2.5.3 Splitting concentration profiles

To split the flow at a node (either 2-way or 3-way), the split profiles are determined by dividing the profile of the inflow proportionally to the velocity ratios of the outflows (see Figure 2.8a) and rescaling appropriately. For example, for two-way splits, if the velocities of the left and right outflows are v_1 and v_2, then the inflow profile is divided into two parts: the left one of width $v_1w/(v_1+v_2)$ and the right one of width $v_2w/(v_1+v_2)$. This produces two profiles which are then rescaled to have width w. Conveniently, splitting a profile represented by an SP-function produces profiles that are also in the same form, and the parameters of these new SP-functions can be determined with a straightforward computation, by rescaling. Thus, in this case no further simplification of the new profiles is needed.

2.5.4 Join-and-split nodes

In the grid, there may also be nodes with two inflows and two outflows. These nodes must have the form shown in Figure 2.8(b), namely the inflow channels are adjacent, and so are the outflow channels. (The other option, with the two inflows being opposite

of each other, is impossible, as this would imply the existence of a flow circulation in the grid.)

We treat this case by reducing it to simple joins and splits, for which we provided solutions earlier. This is done by breaking the computation of the new profile into two sub-steps:

1. Join the two inflow SP-functions, following the method described in Section 2.5.2.

2. Take the computed SP-function profile and split it into two SP-functions for the outflow channels, following the method in Section 2.5.3.

2.6 Experimental results

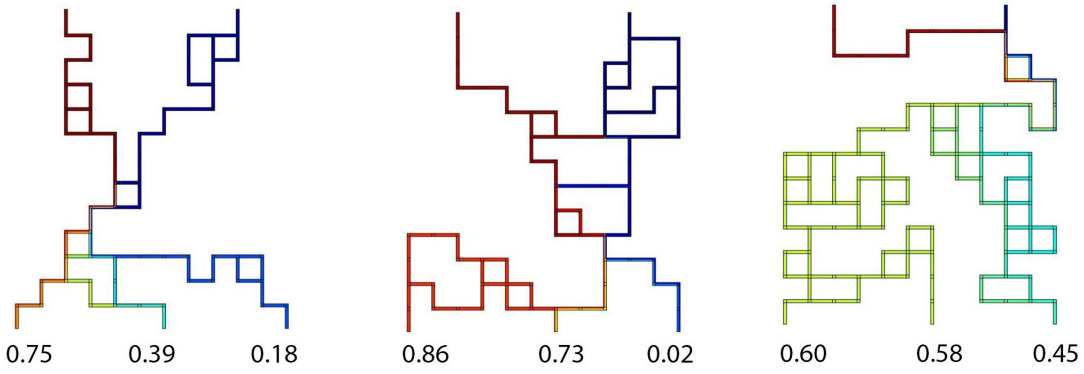


Figure 2.9: Randomly constructed 12×12 grids with concentration values at the outlets.

We used MATLAB® with COMSOL Multiphysics® via LiveLink™ [15] to generate our test grids. Note that in the method from [57], most generated chips have unreachable or "dead-end" channels, namely channels that do not appear on any inlet-to-outlet path.

Such channels are redundant, as they have no effect on the mixing process. Our generator uses a similar approach as in [28] to generate only grids that are connected and have no redundant channels. Figure 2.9 shows three examples of random 12×12 grids obtained from our generator as well as their outlet concentrations.

Our concentration prediction algorithm is implemented in Python and tested on a 3.50GHz quad core 32GB RAM workstation. We conducted an experimental comparison of our algorithm with COMSOL simulation. In our experiment, we used 200 12×12 sample random grids (obtained from our generator) with two inlets (the left has reactant with concentration 1 and the right has buffer with concentration 0) and three outlets. The flow velocities at the inlets are both of constant rate 1 mm/s. The reactant is sodium, with diffusion coefficient value $1.33 \times 10^{-4} mm^2/s$. We used slower velocities than the chips of [57] in order to increase the time the fluid spends in the chip, which improves the variability of concentration values at the outlets.

For COMSOL simulations, we used very fine triangular meshes containing 5-10 million elements. This mesh size was determined through a mesh refinement study to ensure that this setting provides accurate values for outlet concentrations. The basic procedure is to gradually vary the mesh size to determine the size at which the desired parameters have converged to the correct solution. In our case, using a random 12×12 grid, we did a sweep on mesh element size by reducing it at each step and observed the changes in outlet concentrations, repeating until these changes were less than 1% of maximum concentration.

The results for fluid velocities at the 3 outlets are very consistent with COMSOL simulation, with the average percentage difference of velocity values being 0.8%. For concen-

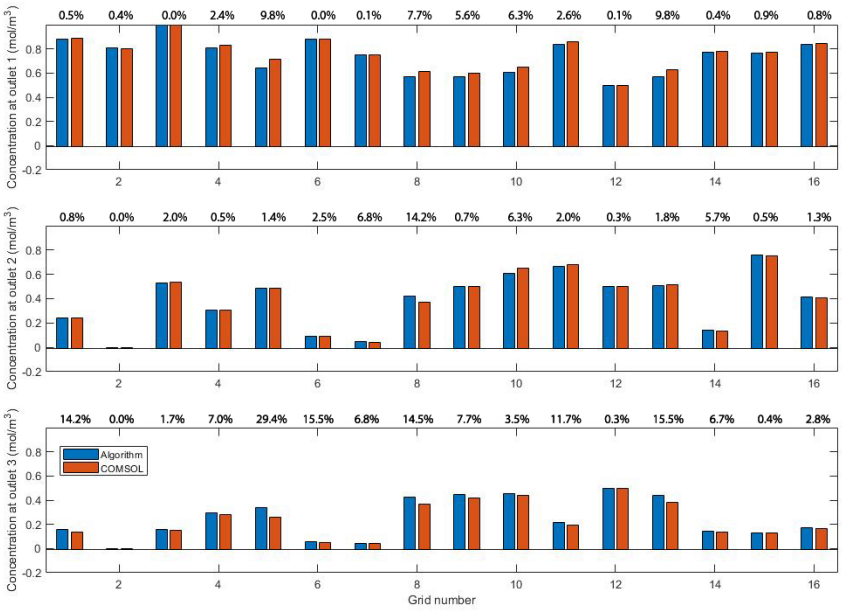


Figure 2.10: Comparison of concentration values at the outlets for sixteen randomly selected grids. Blue bars represent our algorithm and red bars represent COMSOL.

tration values at the outlets, the average absolute difference is 0.006 mol/m^3, which is 0.6% of the maximum concentration. The maximum absolute difference is 0.07 mol/m^3. Figure 2.10 shows the difference of concentration values at the 3 outlets between the algorithm and COMSOL on 16 randomly selected grid designs.

Execution times are measured on the same sample set of 200 grids. On average, with our mesh setting, COMSOL takes approximately 21.3 minutes to finish the computation of one 12×12 grid. These times also vary significantly among different grids, with the fastest time of about 6 minutes and the longest around 30 minutes. Our algorithm is several orders of magnitude faster, requiring on average only 0.0075 second to process one grid. It also uses much less memory: the memory used for concentration profile computations is

only linear in the size of the mixing grid design. COMSOL memory requirements are orders of magnitude higher, as it depends on the size of the mesh used for the simulation.

We also performed an experiment to show that randomly generated 12×12 grid designs can produce a large collection of sufficiently different concentration vectors, and that a useful library of mixing grids can be successfully populated using our random grid generator and concentration prediction algorithm. We generated 50 million random designs on which we ran our algorithm to compute output concentrations. We then filtered out grids that were redundant, in the sense that their output concentration values differed by no more than 0.01. The resulting grid library consisted of 2600 different concentration vectors.

In an additional side experiment, we confirmed that significant reduction in mesh size would not allow us to construct such a database. Reducing the number of mesh elements in COMSOL to the $30,000 - 100,000$ range reduces the simulation time to about 45 seconds — still far too slow for our purpose. It also increases the approximation error to about 10%, which would significantly exceed the 1% granularity of our database, making its content essentially meaningless.

2.7 Discussion

While our current implementation of the algorithm assumes that the input is a grid design, the overall technique applies to arbitrary acyclic planar graphs. The only required assumptions involve inlets and outlets: all inlets and outlets must be located on the external face, cannot interleave, and the inlet concentrations must be non-decreasing in the clockwise

direction along the external face. This is not a significant restriction, as in a microfluidic device its inlets and outlets would normally be located along its external boundary.

Another possible enhancement of our method is related to the assumption that the mixing process in microfluidic grids is exclusively caused by diffusion. This is a valid assumption for flows along straight channels, and it gives a good approximation of outlet concentrations for slow fluid velocity, typical for most microfluidic chips. Prior work in [4] showed, however, that with increased velocities, convection generated in channel bends affects mixing as well. (This will also likely apply to the split and join nodes.) Taking such convections effects into account could further improve the accuracy of our method.

Chapter 3

Convergence of iterative processes for pipe network analysis

In this chapter, we study turbulent flows in large-scale pipe networks, such as water distribution systems and sewage networks. Given the network's topology, the properties of the pipes and fluid, and the nodal inflows/outflows, the objective is to compute flow rates and pressure values within the network. This task involves solving a nonlinear system of governing equations, including the flow conservation equations and the pressure difference equations. The choice of variables in this system leads to different formulations, among which we primarily focus on the loop method. Computing the exact solutions for a pipe network is not feasible, so iterative algorithms that compute approximate solutions have been widely adopted in practice. We focus on the loop-based Newton-Raphson method (NR-loop) and the loop-based Hardy Cross method (HC-loop). We provide a bound on the radius of quadratic convergence for Algorithm NR-loop in Section 3.2. We establish a similar local convergence property for Algorithm HC-loop in Section 3.3; however, in this case, the convergence rate is only linear.

Additionally, in Section 3.4, we explain the errors in [2], and in Section 3.5, we provide an example in which the node-based Newton-Raphson method encounters an infinite loop.

3.1 Preliminaries

Flow graphs. A pipe system can be represented by an undirected graph $G = (V, E)$, where $|V| = n$, $|E| = m$. In G, the pipe junctions are represented by vertices and pipe segments by edges. We assume that G is biconnected, as otherwise its biconnected components can be analyzed separately. For notational reasons, it will be convenient to assign orientations to edges. To this end, we assume that $V = \{1, 2, ..., n\}$, and we define the orientation of an edge e between two different vertices u and v as being from u to v if $u < v$, and from v to u otherwise. For simplicity, the edges in E will be identified by numbers $1, 2, ..., m$. Let D be the $m \times n$ *incident matrix* where $D_{ev} = 1$ if edge e is incident with vertex v and is oriented toward v, $D_{ev} = -1$ if e is oriented away from v, and $D_{ev} = 0$ if e is not incident with v. The specification of a flow graph also involves the *consumption vector* $w \in \mathbb{R}^n$ that satisfies $\sum_{v=1}^{n} w_v = 0$. This vector represents external inflows and outflows, that is the boundary conditions of the flows in the network.

Cycles. A closed loop in the pipe network corresponds to a cycle in its flow graph G. More generally, we define a *cycle* in G as an even-degree subgraph of G. A cycle is called *simple* if it is connected and all of its vertices have degree two. If we view cycles as vectors in $\mathbb{Z}_2^m$ ($\mathbb{Z}_2$ is the finite field of two elements 0 and 1), the set of all cycles in G forms a linear space over $\mathbb{Z}_2$ with vector addition being the operation of symmetric difference. The dimension

of this space is $k = m - n + 1$. A *cycle basis* in G is a collection of k linearly independent simple cycles. Each cycle in G can be obtained as a linear combination of the cycles in the basis. For our analysis, we assume that an arbitrary cycle basis C is provided for the flow graph G. The *total length of* C, denoted ℓ_C, is defined as the sum of the lengths of its cycles.

Each simple cycle in a basis C has two possible orientations, which we refer to as *clockwise* and *counter-clockwise*. These orientations are determined by the two cyclic orderings of vertices in the cycle. The clockwise orientation can be chosen arbitrarily and independently for each cycle in the basis. An edge e on cycle c is called *clockwise on* c if its orientation agrees with the clockwise orientation of c, and otherwise e is called *counter-clockwise on* c. It is worth noting that an edge can be part of multiple cycles and may be clockwise in one cycle while being counter-clockwise in another.

Given a cycle base C, we will number its cycles $1, 2, ..., k$. We can then define the *edge-cycle matrix* A of size $m \times k$ representing C. The value $A_{ec} \in \{-1, 0, 1\}$ indicates the relationship between the orientations of edge e and cycle c. Specifically, it is 1 if edge e is clockwise on cycle c, -1 if e is counter-clockwise, and 0 if e is not part of c.

The flow method. A flow in G is represented by a vector $q \in \mathbb{R}^m$, with q_e denoting the flow on edge e. The sign of q_e indicates the flow direction: $q_e > 0$ iff the flow direction of q agrees with the orientation of e and vice versa. The goal of the flow method is to compute a flow q that satisfies:

$$\begin{cases} D^\top q + w \;=\; 0 & (3.1) \\ A^\top U\, q \;=\; 0 & (3.2) \end{cases}$$

where $U = \text{diag}(|q_e|)$ for $e = 1, 2, \cdots, m$, that is, U is the diagonal matrix whose entries are the absolute values of the flows.

Equation (3.1) is the flow conservation, and Equation (3.2) is the energy conservation. For simplicity, we assume that the coefficient μ_e in the Darcy-Weisbach equations is equal 1 for all edges. However, our derivation can be easily adapted to arbitrary coefficients. With this setup, the pressure difference $\Delta_e p$ along an edge e can be written as $\Delta_e p = |q_e|\, q_e$. Equation (3.2) states that the pressure differences along each cycle need to add up to 0. The energy conservation principle applies to all cycles in G, although for our purposes, it is sufficient to include only the k equations for cycles in the given cycle basis C.

The loop method. The loop method is fundamentally equivalent to the flow method. In the loop method, the flow conservation equations (3.1) are eliminated by the introduction of an arbitrary *reference flow* $\psi \in \mathbb{R}^m$ that already satisfies this equation. Then, the flows in G can be expressed as:

$$q(x) \;=\; \psi + A\, x, \tag{3.3}$$

where $x \in \mathbb{R}^k$ is a variable whose value represents the flow adjustments along cycles in C. Thus, the flow on an edge e is $q_e(x) = \psi_e + \sum_{c=1}^{k} A_{ec} x_c$. In other words, along each cycle c, x_c is added to flows on clockwise edges and subtracted from flows on counter-clockwise edges. In this setting, we define the error function $F : \mathbb{R}^k \mapsto \mathbb{R}^k$ to represent the deviation from the energy conservation equations, and using this function we can express the loop method as:

$$F(x) \;=\; A^\top U(x)\, q(x) \;=\; 0 \tag{3.4}$$

where $U(x)$ is the $m \times m$ diagonal matrix defined above for the flow $q(x)$. That is, if $F(x) = [f_1(x), f_2(x), ..., f_k(x)]^\top$ then $f_c(x) = \sum_{e=1}^{m} A_{ec}\, q_e(x)|q_e(x)| = 0$, for each cycle $c = 1, 2, ..., k$ in the cycle basis C.

We assume that this system has a solution and that the solution is unique. Both existence and uniqueness can be established by using the equivalence to the flow method and reducing the flow method equations to solve a convex optimization problem [51]. In [51], a different proof of uniqueness is also given.

Iterative algorithms. Our focus is on the analysis of solving the nonlinear system (3.4) using two iterative algorithms discussed in the introduction: NR-loop and HC-loop. In general, an iterative algorithm starts with some initial candidate solution $x^{(0)}$, and then it produces successive approximate solutions $x^{(t)}$, attempting to reduce the value of $\|F(x^{(t)})\|$. The process repeats until some stopping criteria are satisfied. Such criteria may involve, for example, an upper bound on the maximum number of iterations, or the improvement being below a desired value. This improvement can be measured either by $\|x^{(t)} - x^{(t-1)}\|$ or $\|F(x^{(t)}) - F(x^{(t-1)})\|$.

An iterative algorithm is called *convergent* if it generates a sequence of values that converges to the actual solution, that is $F(x^{(t)}) \underset{t}{\to} 0$, if it's allowed to run indefinitely. The efficiency of an iterative process can be measured by its rate of convergence, which captures how quickly the generated sequence approaches the solution. The sequence $\left\{x^{(t)}\right\}$ is said to *converge to a solution* x^* *with* ω*-order* [27], for some $\omega \geq 1$, if there exist two constants $\bar{\varepsilon} \geq 0$ and $\bar{t} \geq 1$ such that for all $t \geq \bar{t}$:

$$\|x^{(t)} - x^*\| \leq \bar{\varepsilon}\, \|x^{(t-1)} - x^*\|^{\omega}.$$

For $\omega = 1$ the convergence is said to be linear, and for $\omega = 2$ it is called quadratic.

3.2 Analysis of Newton-Raphson algorithm

The Newton-Raphson algorithm is an iterative process widely used for finding zeros of nonlinear functions. For single-argument functions $\mathbb{R} \to \mathbb{R}$, the algorithm obtains a new approximation value as the intersection of the tangent line to its graph at the current value with the x-axis. This concept can be naturally extended to multi-variate functions $\mathbb{R}^k \to \mathbb{R}^k$ by using the $k \times k$ Jacobian matrix, which represents the first-order partial derivatives.

Given a flow graph G, a cycle basis C of G and a reference flow ψ, we apply the Newton-Raphson algorithm (NR-loop) to solve the system of equations (3.4) starting from some initial solution $x^{(0)}$. (Note that we can as well assume that $x^{(0)} = 0$, for otherwise we can add $x^{(0)}$ to the reference flow and set up the equations for this modified reference flow.) At each step $t \geq 1$, the new solution $x^{(t)}$ is obtained from the previous one using the formula:

$$x^{(t)} \;=\; x^{(t-1)} - F'(x^{(t-1)})^{-1}\, F(x^{(t-1)}) \tag{3.5}$$

where $F'(x)$ denotes the Jacobian of F at x which can be expressed compactly in matrix form as:

$$F'(x) \;=\; 2\, A^{\top}\, U(x)\, A \tag{3.6}$$

It is interesting to note that from the setup, it might seem that the flows $q(x^{(t)})$ obtained by applying the adjustment $x^{(t)}$ would be different for different cycle bases of G. However, upon closer examination, we find that this is not the case. This observation aligns with Nielsen's finding in [39]. We offer a straightforward proof below.

Proposition 2. *Given a flow graph G, a reference flow ψ, and an initial approximation $x^{(0)}$, the flows obtained at each step of Algorithm NR-loop are independent of the choice of cycle basis for G.*

Proof. Let A, B be two edge-cycle matrices that correspond to two different cycle bases of G. Using A and B, set up the two error functions F_A, F_B and let their respective Jacobians be F'_A, F'_B. Denote by $x_A^{(t)}, x_B^{(t)}$ the adjustments at step t acquired from applying Algorithm NR-loop on F_A, F_B respectively. Setting $x_A^{(0)} = x_B^{(0)} = \vec{0}$, we first show that $A\, x_A^{(1)} = B\, x_B^{(1)}$.

We can express $B = AW$ for some $k \times k$ change of basis matrix W. Assuming that $F'_A(0)$ and $F'_B(0)$ are invertible, we have:

$$
\begin{aligned}
B\,x_B^{(1)} &= -B\,F_B'(0)^{-1}\,F_B(0) \\
&= -\tfrac{1}{2}\cdot B\,[B^\top U_B(0)\,B]^{-1}\,B^\top U_B(0)\,\psi \\
&= -\tfrac{1}{2}\cdot A\,W\,[W^\top A^\top U_A(0)AW]^{-1}\,W^\top A^\top U_A(0)\,\psi \\
&= -\tfrac{1}{2}\cdot A\,W\,W^{-1}[A^\top U_A(0)A]^{-1}\,(W^\top)^{-1}\,W^\top A^\top U_A(0)\,\psi \\
&= -\tfrac{1}{2}\cdot A\,[A^\top U_A(0)A]^{-1}\,A^\top U_A(0)\,\psi \\
&= -A\,F_A'(0)^{-1}\,F_A(0) = A\,x_A^{(1)}
\end{aligned}
$$

Since the new flows are the same, we have $U_A(x_A^{(1)}) = U_B(x_B^{(1)})$ which implies that the new Jacobian $F_B'(x_B^{(1)})$ is invertible if and only if $F_A'(x_A^{(1)})$ is invertible. This completes the proof.

□

The convergence of the Newton-Raphson algorithm for nonlinear systems has been studied extensively for decades. The rate of convergence can be as fast as quadratic if the function satisfies certain conditions, for example, when the initial approximation is in close neighborhood of the root and the Jacobians at all successive approximations are invertible. However, the root itself being unknown, it is generally difficult to determine if the initial point is "sufficiently" close to it. The Kantorovich [52] theorem circumvents this challenge by giving the local convergence conditions in terms of the initial point and some general properties of the function. Our statement of this theorem, given below, follows the formulation in [41] but is adapted to functions on Euclidean spaces with one unique root.

For $s \in \mathbb{R}^k$ and $r \in \mathbb{R}$, by $B(s,r)$ we denote the ball centered at s with radius r, that is $B(s,r) = \{x \in \mathbb{R}^k : \|x - s\| \leq r\}$.

Theorem 3 (Kantorovich)**.** *Let $F : \mathbb{R}^k \mapsto \mathbb{R}^k$ be a differentiable function, $F'(x)$ be the $k \times k$ Jacobian matrix of $F(x)$, and $x^{(0)} \in \mathbb{R}^k$ be an initial approximation of the Newton-Raphson process for $F(x) = 0$. Assume that:*

(1) *$F'(x^{(0)})$ is invertible and $\|F'(x^{(0)})^{-1}\| \leq \beta$*

(2) *$\|F'(x^{(0)})^{-1} F(x^{(0)})\| \leq \eta$*

(3) *$\|F'(x) - F'(y)\| \leq L\|x - y\|$ for all $x, y \in \mathbb{R}^k$*

for some $\beta, \eta, L > 0$.

With these assumptions, if $h = \beta\eta L < \frac{1}{2}$ then the sequence $\{x^{(t)}\}$ generated by the Newton-Raphson's iteration process starting at $x^{(0)}$ is well-defined, contained in the ball $B(x^{(0)}, r)$ for $r = (1 - \sqrt{1 - 2h})/(\beta L)$, and converges quadratically to the unique solution x^* of $F(x) = 0$.

Any induced matrix norm $\|\cdot\|$ can be used in this theorem. In our work, we assume the infinity norm on $\mathbb{R}^k$, that is $\|\cdot\| = \|\cdot\|_\infty$. Specifically, the norm of a vector $x \in \mathbb{R}^k$ is $\|x\| = \max_{1 \leq c \leq k} |\, x_c \,|$, and the norm of a matrix $M \in \mathbb{R}^{k \times k}$ is $\|M\| = \max_{1 \leq c \leq k} \sum_{d=1}^{k} |\, M_{cd} \,|$.

Given a flow graph G, we can use Theorem 3 to estimate the radius r of quadratic convergence expressed in terms of properties of F and attributes of G. To this end, we need to estimate the constants in (1), (2) and (3) from Theorem 3. Computing $F(x^{(0)})$ is straightforward, while the inverse of $F'(x^{(0)})$ requires more computation. However, since

no provably accurate and efficient estimates for $\|F'(x)^{-1}\|$ are known, we can use $\beta = \|F'(x^{(0)})^{-1}\|$ and $\eta = \|F'(x^{(0)})^{-1}\, F(x^{(0)})\|$.

Our attention is directed towards the bound (3), which involves the Lipschitz condition on the Jacobian matrix F'. Even though the Newton steps are independent of the choice of cycle basis C, as proven in Proposition 2, the Lipschitz constant may take on different values depending on C. Let $\ell = \ell_C$ be the total length of C. We present a general estimate in terms of ℓ:

Claim 4. $\|F'(x) - F'(y)\| \le 2k\,(\ell - k + 1)\,\|x - y\|$, *for all* $x, y \in \mathbb{R}^k$.

Proof. The derivation for this bound uses the formula (3.6) for the Jacobian of $F(x)$ and (3.3) for the flow values. First, compute:

$$F'(x)_{cd} \;=\; \frac{\partial f_c}{\partial x_d}(x) \;=\; 2\sum_{e=1}^{m} A_{ec}\, A_{ed}\,|\,\psi_e + \sum_{i=1}^{k} A_{ei}\, x_i\,|$$

Then,

$$\begin{aligned}
\|F'(x) - F'(y)\| \;&=\; \max_{1\le c\le k} \sum_{d=1}^{k} \Big|\, \frac{\partial f_c}{\partial x_d}(x) - \frac{\partial f_c}{\partial x_d}(y)\,\Big| \\
&=\; \max_{1\le c\le k} \sum_{d=1}^{k} 2\,\Big|\, \sum_{e=1}^{m} A_{ec}\, A_{ed}\Big(\,|\,\psi_e + \sum_{i=1}^{k} A_{ei}\, x_i\,| - |\,\psi_e + \sum_{i=1}^{k} A_{ei}\, y_i\,|\,\Big)\,\Big| \\
&\le\; \max_{1\le c\le k} \sum_{d=1}^{k} 2 \sum_{e=1}^{m} |\,A_{ec}\,|\,|\,A_{ed}\,|\Big|\, \sum_{i=1}^{k} A_{ei}\,(x_i - y_i)\,\Big| \\
&\le\; 2\, \max_{1\le c\le k} \sum_{d=1}^{k} \sum_{e=1}^{m} |\,A_{ec}\,|\,|\,A_{ed}\,| \sum_{i=1}^{k} |\,A_{ei}\,|\,\|x - y\| \\
&\le\; 2\,(\ell - k + 1)\,k\,\|x - y\|
\end{aligned}$$

because $\sum_{i=1}^{k} |\, A_{ei} \,| \leq k$ and $\max_{1 \leq c \leq k} \sum_{d=1}^{k} \sum_{e=1}^{m} |\, A_{ec} \,|\, |\, A_{ed} \,| \leq \ell - k + 1$, with the last inequality follows from the observation that for each cycle $d \neq c$, at least one of its edges is not on c. □

We are now ready to state sufficient conditions for quadratic convergence of Algorithm NR-loop on the loop equation (3.4). This condition applies to a flow graph $G = (V, E)$, a cycle basis C of G with total length ℓ, a reference flow ψ, and an initial solution $x^{(0)}$ where $F'(x^{(0)})$ is invertible:

Theorem 5 (Convergence conditions of Algorithm NR-loop)**.** *Let* $\beta = \|F'(x^{(0)})^{-1}\|$, $\eta = \|F'(x^{(0)})^{-1}\, F(x^{(0)})\|$, *and* $L = 2k(\ell - k + 1)$. *If* $\beta\eta L < \frac{1}{2}$, *then Algorithm NR-loop converges to the unique solution* x^* *of* $F(x) = 0$ *with quadratic order.*

In what follows, we discuss more specific estimates for L. The estimate of the convergence radius in Theorem 5 critically depends on the choice of the cycle basis. Trivially, for any basis, we have $\ell \leq kn$. Therefore, in Theorem 5, ℓ can be replaced by kn if this bound is sufficient. However, in general, using a cycle basis C with small total length ℓ is likely to improve the convergence radius. The problem of computing cycle bases with small total length has been well studied. In [47], Rizzi introduced an $O(mn)$-time algorithm that generates a cycle basis of total length $\ell = O(m \log n)$, which is within a $2 \log n$ factor of the optimum length. This length bound was further refined to $\ell = O(m \log n / \log(m/n))$ by Kaufmann and Michail, as mentioned in [31, Theorem 4.5]. If the loop equations are built upon the aforementioned cycle basis, the enhanced Lipschitz constant L becomes $2k(\tau m \log n / \log(m/n) - k + 1)$ for some constant τ.

For certain types of graphs, better estimates are possible. In particular, planar graphs emerge naturally when studying flows in pipe networks. If G is planar, the bound on the Lipschitz constant L can be improved by using the face cycle bases in which every edge belongs to at most two cycles (faces):

$$L \;\leq\; 8n. \tag{3.7}$$

Unlike the formula for L in Theorem 5, the proof of (3.7) does not follow from Claim 4 directly. Instead, we argue as follows: For every edge e, $\sum_{i=1}^{k} |\, A_{ei} \,| \leq 2$. Furthermore, for every cycle c, $\sum_{d \neq c} |\, \frac{\partial f_c}{\partial x_d} \,| \leq \frac{\partial f_c}{\partial x_c}$. Thus,

$$\begin{aligned}
\|F'(x) - F'(y)\| \;&\leq\; \max_{1 \leq c \leq k} 2 \Big| \frac{\partial f_c}{\partial x_c}(x) - \frac{\partial f_c}{\partial x_c}(y) \Big| \\
&\leq\; \max_{1 \leq c \leq k} 4 \sum_{e=1}^{m} A_{ec}^2 \Big| \sum_{i=1}^{k} A_{ei} \, (x_i - y_i) \Big| \\
&\leq\; \max_{1 \leq c \leq k} 4 \sum_{e=1}^{m} A_{ec}^2 \sum_{i=1}^{k} |\, A_{ei} \,| \, \|x - y\| \\
&\leq\; 8n \, \|x - y\|
\end{aligned}$$

because $\sum_{e=1}^{m} A_{ec}^2 \leq n$ for any simple cycle c.

Example. We illustrate the local convergence conditions on the simple flow graph depicted in the figure below. The graph is planar and has 4 vertices, 6 edges, 2 inlets and 1 outlet. The cycle basis that is used to set up the loop equations is the cycle basis consisting of internal faces.

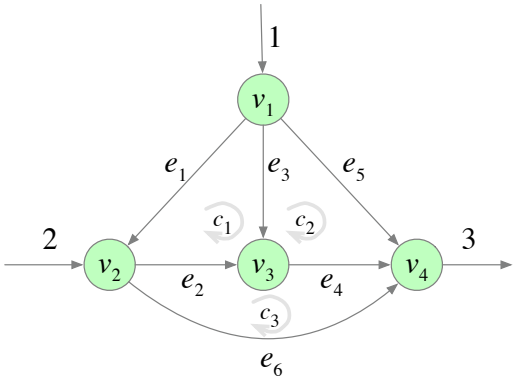


Figure 3.1: An example planar graph where the clockwise directions of the internal face cycles are indicated

The cycle matrix in this example is:

$$\begin{bmatrix} -1 & 0 & 0 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & -1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

Let the reference flow be $\psi = [1, 1, 0, 1, 0, 2]$. The loop equations are:

$$\begin{cases} f_1(x) &= -|1 - x_1|(1 - x_1) - |1 - x_1 + x_3|(1 - x_1 + x_3) + |x_1 - x_2|(x_1 - x_2) \\ f_2(x) &= -|x_1 - x_2|(x_1 - x_2) - |1 - x_2 + x_3|(1 - x_2 + x_3) + |x_2|(x_2) \\ f_3(x) &= |1 - x_1 + x_3|(1 - x_1 + x_3) + |1 - x_2 + x_3|(1 - x_2 + x_3) - |2 - x_3|(2 - x_3) \end{cases}$$

and, the Jacobian is: $F'(x) = [\frac{\partial F}{\partial x_1}(x), \frac{\partial F}{\partial x_2}(x), \frac{\partial F}{\partial x_3}(x)]$, where

$$\frac{\partial F}{\partial x_1}(x) = 2\begin{bmatrix} |1-x_1| + |1-x_1+x_3| + |x_1-x_2| \\ -|x_1-x_2| \\ -|1-x_1+x_3| \end{bmatrix}$$

$$\frac{\partial F}{\partial x_2}(x) = 2\begin{bmatrix} -|x_1-x_2| \\ |x_1-x_2| + |1-x_2+x_3| + |x_2| \\ -|1-x_2+x_3| \end{bmatrix}$$

$$\frac{\partial F}{\partial x_3}(x) = 2\begin{bmatrix} -|1-x_1+x_3| \\ -|1-x_2+x_3| \\ |1-x_1+x_3| + |1-x_2+x_3| + |2-x_3| \end{bmatrix}$$

Let the initial guess be $x^{(0)} = [1.38, 1, 0.93]$. It can be verified that $F'(x^{(0)})$ is invertible. The constants in (1) and (2) are $\|F'(x^{(0)})^{-1}\| < 0.7 = \beta$, $\|F'(x^{(0)})^{-1}\, F(x^{(0)})\| < 0.005 = \eta$ As for the constant in (3), we can set $L = 8n = 32$ since the face cycle basis is used. This gives $\beta\eta L < 0.12$, so the convergence condition is satisfied, and the system converges with quadratic order starting at the given $x^{(0)}$.

3.3 Analysis of Hardy Cross algorithm

In order to simplify the calculation of the new solution $x^{(t)}$ in equation (3.5), a linear operator $H(x)$, typically associated with the Jacobian $F'(x)$, can be used in lieu of it:

$$x^{(t)} \;=\; x^{(t-1)} - H(x^{(t-1)})^{-1}\, F(x^{(t-1)}) \tag{3.8}$$

This variation of the Newton-Raphson method has been thoroughly explored in the literature. In the context of pipe network analysis, Algorithm HC-loop can be viewed as a simplified version of the Newton-Raphson method where the Jacobian matrix F' is replaced by the linear operator H, defined as follows:

$$H(x) \;=\; \mathrm{diag}\Big[\; \frac{\partial f_c}{\partial x_c}(x) \;\Big] \text{ for } c = 1, 2, \cdots, k$$

where $\partial f_c / \partial x_c$ are the diagonal entries of the matrix $F'(x)$, defined in equation (3.6).

As the computation of $H(x)$ is relatively straightforward, Algorithm HC-loop was designed to support manual analysis of small-scale water distribution systems. The process also begins with a reference flow ψ that satisfies the flow conservation equation (3.1). It then iteratively produces new flow adjustments using equation (3.8) until the energy conservation equation (3.2) is satisfied, within some specified tolerance, for all cycles in a given cycle basis C of G.

It is important to note that although equation (3.8) may give the impression that the adjustments for all cycles need to be computed simultaneously, they can be performed for each cycle $c \in C$ separately as follows:

$$x_c^{(t)} \;=\; x_c^{(t-1)} - \Big[\; \frac{\partial f_c}{\partial x_c}(x^{(t-1)}) \;\Big]^{-1} f_c(x^{(t-1)}) \;=\; x_c^{(t-1)} - \frac{\sum\limits_{e=1}^{m} A_{ec}\,|q_e^{(t-1)}|(q_e^{(t-1)})}{2\sum\limits_{e=1}^{m} A_{ec}^2|\,q_e^{(t-1)}\,|}$$

where $q_e^{(t-1)} = \psi_e + \sum_{i=1}^{k} A_{ei}\, x_i^{(t-1)}$ is the flow value on edge e at step $t-1$.

The difference between Algorithm HC-loop and Algorithm NR-loop lies in the fact that in Algorithm NR-loop each adjustment takes into account the interaction be-

tween overlapping cycles, whereas in Algorithm HC-loop these adjustments are computed independently for each cycle.

Local convergence analysis of the variant of the Newton-Raphson algorithm, where a linear operator is used instead of the Jacobian matrix, was studied by Rheinboldt in [46, Theorem 4.3]. Below, we state his results, adapting them to functions in Euclidean spaces with one unique root.

Theorem 6. *Let $F : \mathbb{R}^k \mapsto \mathbb{R}^k$ be a differentiable function, $F'(x)$ be the $k \times k$ Jacobian matrix of $F(x)$, $H : \mathbb{R}^k \mapsto \mathbb{R}^k$ be a linear operator, and $x^{(0)} \in \mathbb{R}^k$ be an initial approximation of the iterative process (3.8) for $F(x) = 0$. Assume that:*

(1) *$H(x^{(0)})$ is invertible and $\|H(x^{(0)})^{-1}\| \leq \beta$*

(2) *$\|H(x^{(0)})^{-1} F(x^{(0)})\| \leq \eta$*

(3) *$\|F'(x) - F'(y)\| \leq L\|x - y\|$ for $x, y \in \mathbb{R}^k$*

(4) *$\|H(x) - H(x^{(0)})\| \leq K\|x - x^{(0)}\|$ for $x \in \mathbb{R}^k$*

(5) *$\|F'(x) - H(x)\| \leq \delta_0 + \delta_1\|x - x^{(0)}\|$ for $x \in \mathbb{R}^k$*

for some $\beta, \eta, L, K > 0$ and $\delta_0, \delta_1 \geq 0$.

With these assumptions, if $\beta\delta_0 < 1$ and $h = \beta\eta L \max(1, (K + \delta_1)/L)/(1 - \beta\delta_0)^2 \leq \frac{1}{2}$, then the sequence $\{x^{(t)}\}$ generated by equation (3.8) starting at $x^{(0)}$ is well-defined, contained in the ball $B(x^{(0)}, r)$ for $r = \eta(1 - \sqrt{1 - 2h})/h(1 - \beta\delta_0)$, and converges linearly to the unique solution x^* of $F(x) = 0$.

We apply Theorem 6 to Algorithm HC-loop for a given flow graph G in a similar manner to our use of Theorem 3 for Algorithm NR-loop. The bound in (3) involving L

remains consistent with those outlined in Section 3.2. For (1), the computation of the inverse of the diagonal matrix $H(x^{(0)})$ is straightforward:

$$\beta = \|H(x^{(0)})^{-1}\| \;=\; \max_{1\le c\le k}\left[\frac{\partial f_c}{\partial x_c}(x^{(0)})\right]^{-1}$$

Similarly, the bound in (2) can be taken as $\eta = \|H(x^{(0)})^{-1}\, F(x^{(0)})\|$. For (4) and (5), the estimates for K, δ_0, and δ_1 depend on the cycle basis C, similar to L. We show the general estimates of these constants in the following claims.

Claim 7. $\|H(x) - H(x^{(0)})\| \le 2(\ell - k + 1)\,\|x - x^{(0)}\|$

The derivation for this bound is as follows:

$$\begin{aligned}
\|H(x) - H(x^{(0)})\| &= \max_{1\le c\le k}\Big|\,\frac{\partial f_c}{\partial x_c}(x) - \frac{\partial f_c}{\partial x_c}(x^{(0)})\,\Big| \\
&= \max_{1\le c\le k} 2\Big|\sum_{e=1}^{m} A_{ec}^2\Big(\,|\,\psi_e + \sum_{i=1}^{k} A_{ei}\,x_i\,| - |\,\psi_e + \sum_{i=1}^{k} A_{ei}\,x_i^{(0)}\,|\,\Big)\Big| \\
&\le \max_{1\le c\le k} 2\sum_{e=1}^{m} A_{ec}^2 \sum_{i=1}^{k} |\,A_{ei}\,|\,\|x - x^{(0)}\| \\
&= 2(\ell - k + 1)\,\|x - x^{(0)}\|
\end{aligned}$$

because $\sum_{e=1}^{m}\sum_{i=1}^{k} A_{ec}^2|\,A_{ei}\,| \le \ell - k + 1$.

Claim 8. $\|F'(x) - H(x)\| \le \delta_0 + \delta_1\,\|x - x^{(0)}\|$ *where* $\delta_0 = 2(\ell - k - 2)\,(\psi_{\max} + k\|x^{(0)}\|)$, $\delta_1 = 2k\,(\ell - k - 2)$, *and* $\psi_{\max} = \max\limits_{1\le e\le m} |\,\psi_e\,|$.

The derivation for this bound is as follows:

$$\begin{aligned}
\|F'(x) - H(x)\| &= \max_{1\le c\le k} \sum_{d\neq c} \Big| \frac{\partial f_c}{\partial x_d}(x) \Big| \\
&= \max_{1\le c\le k} \sum_{d\neq c} 2 \Big| \sum_{e=1}^{m} A_{ec}\, A_{ed}\, |\, \psi_e + \sum_{i=1}^{k} A_{ei}\, x_i\, | \Big| \\
&\le \max_{1\le c\le k} 2 \sum_{d\neq c} \sum_{e=1}^{m} |\, A_{ec}\, |\, |\, A_{ed}\, | \big(\psi_{\max} + k\|x\| \big) \\
&\le 2(\ell - k - 2) \big(\psi_{\max} + k\|x^{(0)}\| + k\|x - x^{(0)}\| \big) \\
&\le 2(\ell - k - 2) \left(\psi_{\max} + k\|x^{(0)}\|\right) + 2k\,(\ell - k - 2)\, \|x - x^{(0)}\|
\end{aligned}$$

because $\sum_{d\neq c} \sum_{e=1}^{m} |\, A_{ec}\, |\, |\, A_{ed}\, | \le \ell - k + 1 - 3 = \ell - k - 2$. (This uses the fact that each cycle c has at least 3 edges.)

The following theorem states the sufficient condition for linear convergence of Algorithm HC-loop on loop equation $F : \mathbb{R}^k \mapsto \mathbb{R}^k$, for a flow graph $G = (V, E)$ with a given cycle basis C of total length ℓ, a reference flow ψ with the maximum absolute flow value flow $\psi_{\max}$, and an initial solution $x^{(0)}$ for which $H(x^{(0)})$ is invertible:

Theorem 9 (Convergence conditions of Algorithm HC-loop)**.** *Let* $\beta = \|H(x^{(0)})^{-1}\|$, $\eta = \|H(x^{(0)})^{-1}\, F(x^{(0)}\|$, $K = 2(\ell - k + 1)$, $L = 2k\,(\ell - k + 1)$, $\delta_0 = 2(\ell - k - 2)\left(\psi_{\max} + k\|x^{(0)}\|\right)$, *and* $\delta_1 = 2k\,(\ell - k - 2)$. *If* $\beta\delta_0 < 1$ *and* $\beta\eta L \max(1, (K + \delta_1)/L)/(1 - \beta\delta_0)^2 \le \frac{1}{2}$, *then Algorithm HC-loop converges to the unique solution* x^* *of* $F(x) = 0$ *with linear order.*

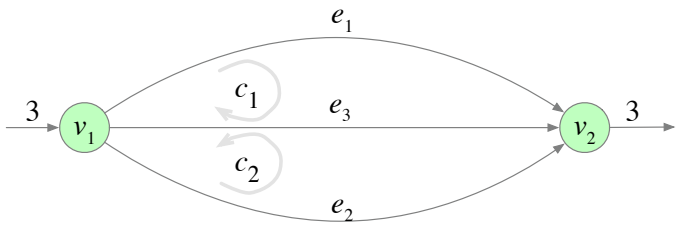
We remark, similar to Section 3.2, more specific estimates of the constants K, δ_0 and δ_1 can be obtained by substituting the value for ℓ corresponding to the cycle basis used in the algorithm.

In the case where G is planar and the face cycle basis is used, the constant K can be improved to $4n$. This bound can be derived as follows:

$$\begin{aligned}
\|H(x) - H(x^{(0)})\| &\leq \max_{1\leq c\leq k} 2 \sum_{e=1}^{m} A_{ec}^2 \sum_{i=1}^{k} |\, A_{ei} \,| \, \|x - x^{(0)}\| \\
&\leq \max_{1\leq c\leq k} 2 \sum_{e=1}^{m} A_{ec}^2 \cdot 2 \, \|x - x^{(0)}\| \\
&\leq 4n \, \|x - x^{(0)}\|
\end{aligned}$$

Furthermore, utilizing the property that for any edge e, $\sum_{i=1}^{k} |\, A_{ei} \,| \leq 2$, the constants δ_0 and δ_1 can also be improved to $2(\ell - k - 2)\big(\, \psi_{\max} + 2\|x^{(0)}\| \,\big)$ and $4(\ell - k - 2)$, respectively.

Lower bound on convergence. We now demonstrate that, in general, the guaranteed local convergence rate of Algorithm HC-loop is not better than linear. To illustrate this, consider the flow graph G with 2 vertices, 3 edges, 1 inlet and 1 outlet. The cycle basis consists of two cycles, whose clockwise directions are shown in the figure below.



Let the reference flow be $\psi = [0, 0, 3]$. The loop equations are:

$$\begin{aligned} f_1(x) &= |x_1|(x_1) - |3 - x_1 - x_2|(3 - x_1 - x_2) \\ f_2(x) &= |x_2|(x_2) - |3 - x_1 - x_2|(3 - x_1 - x_2) \end{aligned}$$

The solution is $x_1^* = x_2^* = 1$. We are interested in the case where the initial adjustment $x^{(0)}$ is sufficiently close to the solution, particularly when $x_1^{(0)} = x_2^{(0)} = 1 \pm \epsilon$ for some $0 < \epsilon \ll 1$. The first adjustment is:

$$x_1^{(1)} = x_2^{(1)} = 1 \pm \epsilon \mp \frac{3\epsilon\,(2 \pm \epsilon)}{2(2 \pm \epsilon)} = 1 \pm \epsilon/2$$

Thus, $\| x^{(1)} - x^* \| = \epsilon/2 = \frac{1}{2} \| x^{(0)} - x^* \|$. The convergence rate is exactly $\frac{1}{2}$.

3.4 Errors in [2]

Altman and Boulos [2] attempted to provide local convergence conditions of Algorithm NR-loop and Algorithm NR-flow for solving flow equations. However, their analysis has two errors that make their bounds on convergence invalid. The first one is an algebraic mistake when substituting the variables in [2, eq. (13)]:

$$\begin{aligned} x_1 &= \eta_{11} x_{t+1} + \eta_{12} x_{t+2} + \cdots + \eta_{1t} x_e + a_1 \\ &\vdots \\ x_t &= \eta_{t1} x_{t+1} + \eta_{t2} x_{t+2} + \cdots + \eta_{tt} x_e + a_t \end{aligned}$$

into [2, eq. (12)]:

$$\begin{aligned} \gamma_{11}\xi_1 x_1^2 + \gamma_{12}\xi_2 x_2^2 + \cdots + \gamma_{1e}\xi_e x_e^2 + \Phi_1 &= 0 \\ &\vdots \\ \gamma_{m1}\xi_1 x_1^2 + \gamma_{m2}\xi_2 x_2^2 + \cdots + \gamma_{me}\xi_e x_e^2 + \Phi_m &= 0 \end{aligned}$$

(The indices in the last equation in [2] were also incorrectly given as n instead of m. We corrected these above.) With a change of variables from $x_{t+1}, x_{t+2}, \cdots, x_e$ to $x_1, x_2, \cdots, x_l$, they obtained [2, eq. (14)]:

$$\begin{aligned} f_1 &= p_{11}x_1^2 + p_{12}x_2^2 + \cdots + p_{1l}x_l^2 + b_{11}x_1 + b_{12}x_2 + \cdots + b_{1l}x_l + d_1 = 0 \\ &\vdots \\ f_l &= p_{l1}x_1^2 + p_{l2}x_2^2 + \cdots + p_{1l}x_l^2 + b_{l1}x_1 + b_{l2}x_2 + \cdots + b_{ll}x_l + d_l = 0 \end{aligned}$$

All the terms $x_i x_j$ where $i \neq j$ do not appear in the above equations which affect the computation of the partial derivatives $\partial f_i / \partial x_j$ later on in their analysis.

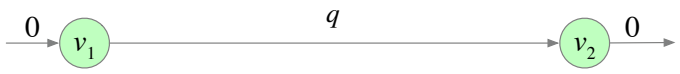
The other mistake involves the sign issue in the energy equation for each cycle. The authors used a fixed indicator γ_{wu} for the orientation of flow in cycle w in [2, eq. (6)]:

$$\Phi_w + \sum_{u=1}^{e} \gamma_{wu}\xi_u Q_u^2 = 0$$

This would be true if all flow directions remained the same, in other words, if the sign of each Q_u was constant through the whole iterative process. However, it is not guaranteed.

Thus, when Q_u flips sign, γ_{wu} needs to be changed accordingly for the energy equation to be valid.

3.5 An example of non-convergence of algorithm NR-node



Shamir mentioned the issue with non-convergence encountered in certain systems in [49]. We provide a simple flow graph in which Algorithm NR-node enters an infinite loop. Using similar setup for the node method as in [49], the unknowns are pressure p_v at node v for $v = 1, 2, \cdots, n$, and the system of equations includes flow conservation equations for each node of the form:

$$f_v(p_1, p_2, \cdots, p_n) = w_v + \sum_{e=(u,v)\in E} D_{ev}\, q_e = w_v + \sum_{e=(u,v)\in E} D_{ev} \frac{p_u - p_v}{\sqrt{|p_u - p_v|}} \;=\; 0 \qquad (3.9)$$

The below example demonstrates an oscillation case of Algorithm HR-node for solving the described node method equation. Consider the simple flow graph G with only two vertices v_1 and v_2 connected by a single edge as shown in the figure below. Set the external flow on the two vertices to be 0 and the corresponding pressures to be $p_1 = x \neq 0$ and $p_2 = 0$.

The equation (3.9) is then $f(x) = \frac{-x}{\sqrt{|x|}} = 0$. The iteration step is: $x - \frac{f(x)}{f'(x)} = x - \frac{x \cdot 2\sqrt{|x|}}{\sqrt{|x|}} = -x$. So for any initial value $x \neq 0$, the algorithm will oscillate between x and $-x$.

Chapter 4

Hardness results for the minimum spanning tree congestion problem

In this chapter, we study the minimum spanning tree congestion problem (STC). In Section 4.2, we address an open problem in [43] by providing an improved hardness result for K-STC (STC for a fixed integer K). We prove that K-STC is $\mathbb{NP}$-complete for $K \geq 5$, leaving only 4-STC open.

In Section 4.3, we explore K-STC for bipartite graphs with a radius of 2. We establish that for $K \geq 6$, K-STC remains $\mathbb{NP}$-complete for this class of graphs. In Section 4.4, we consider K-STCD, a variant of STC in which the objective is to determine if the graph has a spanning tree of depth at most D and congestion at most K. We establish a tight bound for bipartite graphs by providing an $\mathbb{NP}$-completeness proof for 6-STC2 and a polynomial-time algorithm for 5-STC2. Additionally, we present two polynomial-time algorithms for STC2 for bipartite graphs with restrictions on vertex degrees.

4.1 Preliminaries

Basic graph terminology. Let G be a simple graph with vertex set V and edge set E. We use notation $N_G(v)$ for the neighborhood of a vertex $v \in V$ and v for its degree. For a vertex $v \in V$, its *eccentricity* $\mathrm{ecc}_G(v)$ is defined as the maximum distance from v to any other vertex. The *radius* of G is $\mathrm{rad}(G) = \min_{v \in V} \mathrm{ecc}_G(v)$.

Consider a spanning tree $T \subseteq E$ of G. If $e = (u, v) \in T$, removing e from T splits T into two subtrees. We denote by $T_{u,v}$ the subtree that contains u and by $T_{v,u}$ the subtree that contains v. Let the *cut-set* of e, denoted $\partial_{G,T}(e)$, be the set of edges in E that have one endpoint in $T_{u,v}$ and the other in $T_{v,u}$. In other words, $\partial_{G,T}(e)$ consists of the edges $(u', v') \in E$ for which the unique (simple) path in T from u' to v' goes through e. Note that $e \in \partial_{G,T}(e)$. The *congestion of* e, denoted by $\mathrm{cng}_{G,T}(e)$, is the cardinality of $\partial_{G,T}(e)$. The *congestion of tree* T is $\mathrm{cng}_G(T) = \max_{e \in T} \mathrm{cng}_{G,T}(e)$. Finally, the *spanning tree congestion of graph* G, denoted by $\mathrm{stc}(G)$, is defined as the minimum value of $\mathrm{cng}_G(T)$ over all spanning trees T of G.

Weighted edges. The concept of the spanning tree congestion extends naturally to edge-weighted graphs. An edge e with integer weight $\omega \geq 1$ contributes ω to the congestion of any edge f for which $e \in \partial_{G,T}(f)$. One can think of e as representing ω parallel edges between u and v. We refer to these parallel edges as a *non-weighted realization* of a weighed edge e. Indeed, replacing e by this realization does not affect the minimum congestion value, because in a multigraph only one edge between any two given vertices can be in a spanning tree, but all of them belong to the cut-set $\partial_{G,T}(f)$ of any edge $f \in T$ whose removal separates these vertices in T (and thus all contribute to $\mathrm{cng}_{G,T}(f)$).

We can also realize a weighted edge using a simple graph (without multiple edges). As observed in [44] (and is easy to prove), edge subdivision does not affect the spanning tree congestion of a graph, so instead of using parallel edges we can realize an edge of weight ω using ω parallel disjoint paths. (See Figure 4.1 for illustration.) We state our results in terms of simple graphs, but we use weighted graphs in our proofs with the understanding that they actually represent simple graphs. As all weights used in the paper are constant, the computational complexity of K-STC is not affected. The proof in Section 4.2 does not depend on what realization of weighted edges we use, while the proof in Section 4.3 uses a specific realization that we refer to as *spintop*: an edge (u, v) of weight ω is realized using $\omega - 1$ length-three u-to-v paths in addition to a non-weighted edge (u, v) itself (see Figure 4.1b).

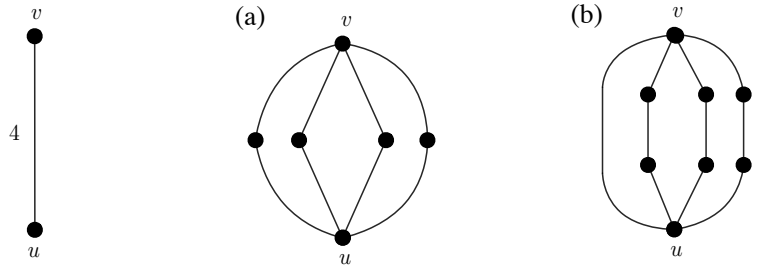


Figure 4.1: Two different realizations of an edge (u, v) of multiplicity 4. (a) A basic realization using paths of length 2. (b) The spintop realization used in Section 4.3.

Double weights. In fact, it is convenient to generalize this further by introducing edges with *double weights*. A double weight of an edge e is denoted $\omega : \omega'$, where ω and ω' are positive integers such that $\omega \leq \omega' \leq K - 1$, and its interpretation in the context of K-STC is as follows:

Given a spanning tree T,

- if $e \in E \setminus T$ then e contributes ω to the congestion $\mathrm{cng}_{G,T}(f)$ of any edge $f \in T$ for which $e \in \partial_{G,T}(f)$, and
- if $e \in T$ then e contributes ω' to its own congestion, $\mathrm{cng}_{G,T}(e)$.

The lemma below provides a simple-graph realization of double-weighted edges. It implies that including such edges does not affect the computational complexity of K-STC, allowing us to formulate our proofs in terms of graphs where some edges have double weights.

Lemma 10. *Let (u, v) be an edge in G with double weight $\omega : \omega'$, where $\omega \leq \omega' \leq K - 1$. Consider another graph G' obtained from G by removing (u, v), and for each $i = 1, 2, \cdots, \omega$ adding a new vertex w_i with two edges: edge (u, w_i) of weight 1 and edge (w_i, v) of weight $\omega' - \omega + 1$ (see Figure 4.2a for an example). Then, $stc(G) \leq K$ if and only if $stc(G') \leq K$.*

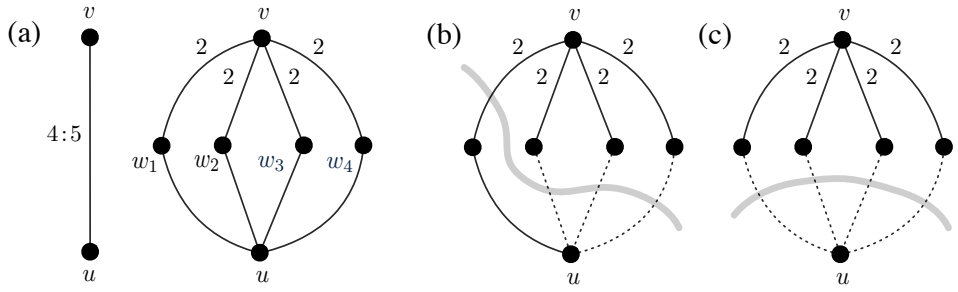


Figure 4.2: (a) On the left, an edge (u, v) with double weight $4:5$ in G. On the right, the realization of (u, v) in G'. If one applies the spintop realization of the edges from v to w_i's, as in Figure 4.1b, then the subgraph on the right realizing (u, v) is bipartite and all its nodes are within distance 2 from v. Figures (b) and (c) illustrate the proof of Lemma 10: (b) the traversal of T' and the cut of (u, v) when $(u, v) \in T$, (c) the traversal of T' and the cut containing (u, v) when $(u, v) \notin T$. Solid lines are tree edges and dotted lines are non-tree edges.

Proof. Denote by $W = \{w_1, w_2, ..., w_\omega\}$ the set of new vertices, and by $W_u = \{(u, w_i) \mid w_i \in W\}$ and $W_v = \{(w_i, v) \mid w_i \in W\}$ the sets of new edges added to G'.

($\Rightarrow$) Suppose that G has a spanning tree T with $\text{cng}_G(T) \leq K$. We will show that there exists a spanning tree T' of G' with $\text{cng}_{G'}(T') \leq K$. We break the proof into two cases, in both cases showing that $\text{cng}_{G',T'}(e) \leq K$ for each edge $e \in T'$.

Case 1: $(u, v) \in T$.

Consider the spanning tree $T' = T \setminus \{(u, v)\} \cup W_v \cup \{(w_1, u)\}$ of G' (see Figure 4.2b). For every edge $(x, y) \in E \setminus \{(u, v)\}$, the x-to-y paths in T and T' are the same, except that if the x-to-y path in T traverses edge (u, v) then the x-to-y path in T' traverses $(u, w_1), (w_1, v)$ instead. Therefore,

- If $e \in T' \setminus (W_v \cup \{(u, w_1)\})$, then $\partial_{G',T'}(e) = \partial_{G,T}(e)$. So $\text{cng}_{G',T'}(e) = \text{cng}_{G,T}(e) \leq K$.
- If $e = (u, w_1)$, then $\partial_{G',T'}(e) = \partial_{G,T}(u, v) \setminus \{(u, v)\} \cup W_u$. By the definition of double weights, (u, v) contributes ω' to $\text{cng}_{G,T}(u, v)$ while each edge in W_u contributes 1 to $\text{cng}_{G',T'}(e)$. Hence, $\text{cng}_{G',T'}(e) = \text{cng}_{G,T}(u, v) - \omega' + \omega \leq \text{cng}_{G,T}(e) \leq K$.
- If $e = (w_1, v)$, then $\partial_{G',T'}(e) = \partial_{G,T}(u, v) \setminus \{(u, v)\} \cup \{e\} \cup (W_u \setminus \{(w_1, u)\})$. Since e contributes $\omega' - \omega + 1$ to its own congestion, we have: $\text{cng}_{G',T'}(e) = \text{cng}_{G,T}(u, v) - \omega' + (\omega' - \omega + 1) + (\omega - 1) = \text{cng}_{G,T}(u, v) \leq K$.
- Lastly, if $e \in W_v \setminus \{(w_1, v)\}$ then it is a leaf edge, we have $\text{cng}_{G',T'}(e) = \omega' - \omega + 2 \leq \omega' + 1 \leq K$.

Case 2: $(u, v) \notin T$.

Let $T' = T \cup W_v$, which is a spanning tree of G' (see Figure 4.2c). We consider the following sub-cases:

- If $e \in W_v$ then, as e is a leaf edge, we have $\text{cng}_{G',T'}(e) = \omega' - \omega + 2 \leq \omega' + 1 \leq K$.
- If $e \in T' \setminus W_v$ and e is not on the u-to-v path in T', then $\partial_{G',T'}(e) = \partial_{G,T}(e)$. So $\text{cng}_{G',T'}(e) = \text{cng}_{G,T}(e) \leq K$.
- If $e \in T' \setminus W_v$ and e is on the u-to-v path in T', then $\partial_{G',T'}(e) = \partial_{G,T}(e) \setminus \{(u, v)\} \cup W_u$. Since (u, v) contributes ω to $\text{cng}_{G,T}(e)$ and W_u also contributes ω to $\text{cng}_{G',T'}(e)$, we have $\partial_{G',T'}(e) = \partial_{G,T}(e) \leq K$.

We have shown that $\text{cng}_{G'}(T') \leq K$ in all cases, which completes the proof for the forward implication. We now proceed to the proof of the converse implication.

($\Leftarrow$) Let T' be the spanning tree of G' with congestion $\text{cng}_{G'}(T') \leq K$. We will show that there exists a spanning tree T of G with $\text{cng}_G(T) \leq K$. Note that, for any $w_i \in W$, T' traverses at least one of the two edges (u, w_i) and (w_i, v). Furthermore, at most one vertex in W is a non-leaf. We consider three cases. In the first two cases the arguments follow the same pattern as in the proof for the ($\Rightarrow$) implication, in essence reversing the modification of the spanning tree. Then the third case reduces to the second case.

Case 1: Exactly one vertex in W is a non-leaf in T'.

Without loss of generality, we can assume w_1 is a non-leaf vertex (that is, both (u, w_1) and (w_1, v) are in T) and $W \setminus \{w_1\}$ are leaves. We construct T by adding (u, v) to T' and removing all vertices of W and their incident edges from T. By the construction, T is a spanning tree of G. We have:

- If $e \in T \setminus \{(u,v)\}$, then $\text{cng}_{G,T}(e) = \text{cng}_{G',T'}(e) \leq K$.

- If $e = (u,v)$, then $\text{cng}_{G,T}(e) \leq \text{cng}_{G',T'}(v, w_1) \leq K$.

Case 2: All vertices in W are leaves and T' traverse all edges in W_v.

Let $T = T' \setminus W_v$, which is a spanning tree of G. Then

- If $e \in T$ and e is not on the u-to-v path in T, then $\text{cng}_{G,T}(e) = \text{cng}_{G',T'}(e) \leq K$.

- If $e \in T$ and e is on the u-to-v path in T, then (u,v) and W_u contribute the same amount ω to the congestion of e in T and T', respectively, implying that $\text{cng}_{G,T}(e) = \text{cng}_{G',T'}(e) \leq K$.

Case 3: All vertices in W are leaves and T' traverses at least one edge in W_u.

In this case, we consider another spanning tree T'' of G' that traverses all edges in W_v and does not use any edge in W_u. It is sufficient to show that $\text{cng}_{G'}(T'') \leq \text{cng}_{G'}(T')$, since it implies that $\text{cng}_{G'}(T'') \leq K$, and then we can apply Case 2 to T''. We examine the congestion values of each edge $e \in T''$:

- If $e \in T'' \setminus W_v$ and e is not on the u-to-v path in T'', then $e \in T'$ and $\partial_{G',T''}(e) = \partial_{G',T'}(e)$, implying $\text{cng}_{G',T''}(e) = \text{cng}_{G',T'}(e)$.

- If $e \in T'' \setminus W_v$ and e is on the u-to-v path in T'', then for each vertex $w_i \in W$ either (u, w_i) contributes 1 or (w_i, v) contributes $\omega' - \omega + 1 \geq 1$ to $\text{cng}_{G',T'}(e)$. On the other hand, in T'', all edges in W_u are in $\partial_{G',T''}(e)$ and contribute a total of ω to $\text{cng}_{G',T''}(e)$. Thus, $\text{cng}_{G',T''}(e) \leq \text{cng}_{G',T'}(e)$.

- If $e \in W_v$, then $\text{cng}_{G',T''}(e) = \omega' - \omega + 2 \leq \omega' + 1 \leq K$.

In all cases, we have proved that there is a spanning tree T of G that has congestion at most K establishing the validity of the backward implication. □

As explained earlier, in Section 4.3 we will use the spintop realization for weighted edges. With this, the realization of an edge $e = (u, v)$ with double weight $\omega{:}\omega'$ will use the spintop realization for the edges of weight $\omega' - \omega + 1$ between v and the w_i's. The property of this realization that will be crucial in Section 4.3 is that it is bipartite and all its nodes are within distance 2 from v.

Remark. There is a simpler way to realize an edge (u, v) with a double weight $\omega{:}\omega'$: replace it by a length-2 path (u, w), (w, v), where w is a new vertex, edge (u, w) has weight ω, and edge (w, v) has weight ω'. This indeed works, but can be used only when $\omega + \omega' \leq K$. This is because, in this construction, if w is a leaf of a spanning tree, the congestion of the tree edge from w will be $\omega + \omega'$, and this congestion value cannot exceed K. This realization of double-weighted edges would suffice for our proof in Section 4.2, but not the one in Section 4.3.

4.2 $\mathbb{NP}$-completeness proof of K-STC for $K \geq 5$

In this section, we present the following hardness results for K-STC:

Theorem 11. *For any fixed integer $K \geq 5$, K-STC is $\mathbb{NP}$-complete.*

Our proof uses an $\mathbb{NP}$-complete variant of the satisfiability problem called (2P1N)-SAT [17, 60]. An instance of (2P1N)-SAT is a boolean expression ϕ in conjunctive normal form, where each variable occurs exactly three times, twice positively and once negatively,

and each clause contains exactly two or three literals of different variables. The objective is to decide if ϕ is satisfiable, that is if there is a satisfying assignment that makes ϕ true.

For each constant K, K-STC is clearly in $\mathbb{NP}$. We will present a polynomial-time reduction from (2P1N)-SAT. In this reduction, given an instance ϕ of (2P1N)-SAT, we construct a graph G with the following property:

($*$) ϕ has a satisfying truth assignment if and only if $\text{stc}(G) \leq K$.

Throughout the proof, the three literals of x_i in ϕ will be denoted by x_i, x_i', and $\bar{x}_i$, where x_i, x_i' are the two positive occurrences of x_i and $\bar{x}_i$ is the negative occurrence of x_i. We will also use notation $\tilde{x}_i$ to refer to an unspecified literal of x_i, that is $\tilde{x}_i \in \{x_i, x_i', \bar{x}_i\}$.

We now describe the reduction. Set $k_i = K - i$ for $i = 1, 2, 3, 4$. (In particular, for $K = 5$, we have $k_1 = 4$, $k_2 = 3$, $k_3 = 2$, $k_4 = 1$). G will consist of gadgets corresponding to variables, with the gadget corresponding to x_i having three vertices x_i, x_i', and $\bar{x}_i$, that represent its three occurrences in the clauses. G will also have vertices representing clauses and edges connecting literals with the clauses where they occur (see Figure 4.3b for an example). As explained in Section 4.1, without any loss of generality, we can allow edges in G to have constant-valued weights, single or double. Specifically, starting with G empty, the construction of G proceeds as follows:

- Add a *root vertex* r.
- For each variable x_i, construct the *x_i-gadget* (see Figure 4.3a). This gadget has three vertices corresponding to the literals: a *negative literal vertex* $\bar{x}_i$ and two *positive literal vertices* x_i, x_i', and two auxiliary vertices y_i and z_i. Its edges and their weights are given in the table below:

edge	$(\bar{x}_i, z_i)$	(z_i, x_i)	(x_i, x'_i)	(r, x'_i)	(r, y_i)	(y_i, z_i)	$(y_i, \bar{x}_i)$
weight	$1:k_3$	$1:k_3$	$1:k_2$	k_3	k_4	k_4	$1:k_2$

- For each clause c, create a *clause vertex* c. For each literal $\tilde{x}_i$ in c, add the corresponding *clause-to-literal edge* $(c, \tilde{x}_i)$ of weight $1:k_2$. Importantly, as all literals in c correspond to different variables, these edges will go to different variable gadgets.

- For each two-literal clause c, add a *root-to-clause* edge (r, c) of weight $1:k_1$.

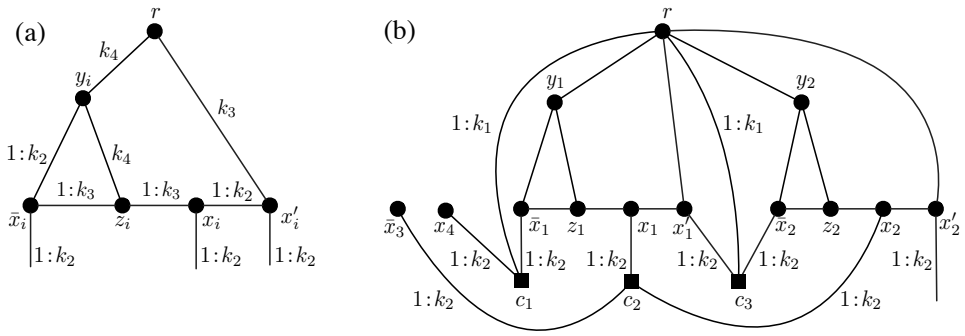


Figure 4.3: (a)The x_i-gadget. (b) An example of a partial graph G for the boolean expression $\phi = c_1 \wedge c_2 \wedge c_3 \wedge \cdots$ where $c_1 = \bar{x}_1 \vee x_4$, $c_2 = x_1 \vee x_2 \vee \bar{x}_3$, and $c_3 = x_1 \vee \bar{x}_2$. (The weights of edges inside the variable gadgets are not shown.)

We now show that G has the required property $(*)$, proving the two implications separately.

($\Rightarrow$) Suppose that ϕ has a satisfying assignment. Using this assignment, we construct a spanning tree T of G as follows:

- For every x_i-gadget, include in T edges (r, x'_i), (r, y_i), and (y_i, z_i). If $x_i = 0$, include in T edges $(\bar{x}_i, z_i)$ and (x_i, x'_i), otherwise include in T edges $(y_i, \bar{x}_i)$ and (z_i, x_i).

- For each clause c, include in T one clause-to-literal edge that is incident to any literal vertex that satisfies c in our chosen truth assignment for ϕ.

By routine inspection, T is indeed a spanning tree of G: Each x_i-gadget is traversed from r without cycles, and all clause vertices are leaves of T. Figures 4.4 and 4.5 show how T traverses an x_i-gadget in different cases, depending on whether $x_i = 0$ or $x_i = 1$ in the truth assignment for ϕ, and on which literals are chosen to satisfy each clause. Note that the edges with double weights satisfy the assumption of Lemma 10 in Section 4.1, that is each such weight $1{:}\omega'$ satisfies $1 \leq \omega' \leq K - 1$.

We need to verify that each edge in T has congestion at most K. All the clause vertices are leaves in T, thus the congestion of each clause-to-literal edge is $k_2 + 2 = K$ (this holds for both three-literal and two-literal clauses). To analyze the congestion of the edges inside an x_i-gadget, we consider two cases, depending on the value of x_i in our truth assignment.

When $x_i = 0$, we have two sub-cases (a) and (b) as shown in Figure 4.4. The congestions of the edges in the x_i-gadget are as follows:

- In both cases, $\mathrm{cng}_{G,T}(r, x_i') = k_3 + 3$.
- In case (a), $\mathrm{cng}_{G,T}(r, y_i) = k_4 + 3$. In case (b), it is $k_4 + 2$.
- In case (a), $\mathrm{cng}_{G,T}(y_i, z_i) = k_4 + 4$. In case (b), it is $k_4 + 3$.
- In case (a), $\mathrm{cng}_{G,T}(\bar{x}_i, z_i) = k_3 + 3$. In case (b), it is $k_3 + 2$.
- In both cases, $\mathrm{cng}_{G,T}(x_i, x_i') = k_2 + 2$.

On the other hand, when $x_i = 1$, we have four sub-cases. Figure 4.4 illustrates cases (a)–(c). In case (d) (not shown in Figure 4.4), none of the positive literal vertices

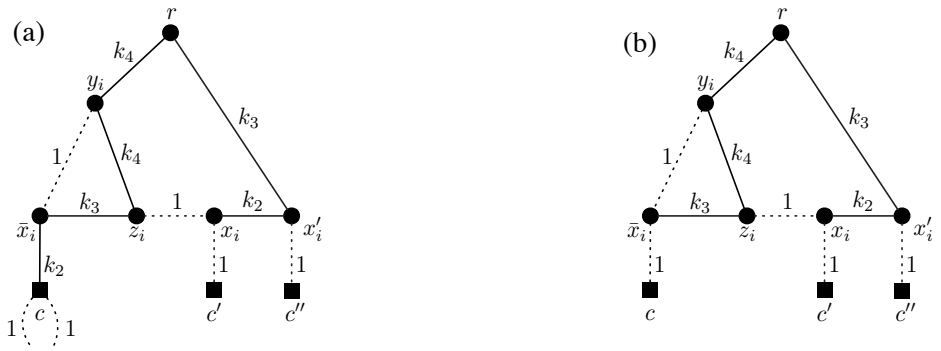


Figure 4.4: The traversal of the x_i-gadget by T when $x_i = 0$. Solid lines are tree edges, dotted lines are non-tree edges. (a) $\bar{x}_i$ is chosen by clause c. (b) $\bar{x}_i$ is not chosen by clause c.

x_i, x'_i is chosen to satisfy their corresponding clauses. The congestions of the edges in the x_i-gadget are as follows:

- In cases (a) and (b), $\text{cng}_{G,T}(r, x'_i) = k_3 + 3$. In cases (c) and (d), it is $k_3 + 2$.
- In cases (a) and (c), $\text{cng}_{G,T}(r, y_i) = k_4 + 4$. In cases (b) and (d), it is $k_4 + 3$.
- In cases (a) and (c), $\text{cng}_{G,T}(y_i, z_i) = k_4 + 4$. In cases (b) and (d), it is $k_4 + 3$.
- In cases (a) and (c), $\text{cng}_{G,T}(z_i, x_i) = k_3 + 3$. In cases (b) and (d), it is $k_3 + 2$.
- In all cases, $\text{cng}_{G,T}(y_i, \bar{x}_i) = k_2 + 2$.

In summary, the congestion of each edge of T is at most K. Thus $\text{cng}_G(T) \leq K$; in turn, $\text{stc}(G) \leq K$, as claimed.

($\Leftarrow$) We now prove the other implication in ($*$). We assume that G has a spanning tree T with $\text{cng}_G(T) \leq K$. We will show how to convert T into a satisfying truth assignment for ϕ. The proof consists of a sequence of claims showing that T must have a special form that will allow us to define this truth assignment.

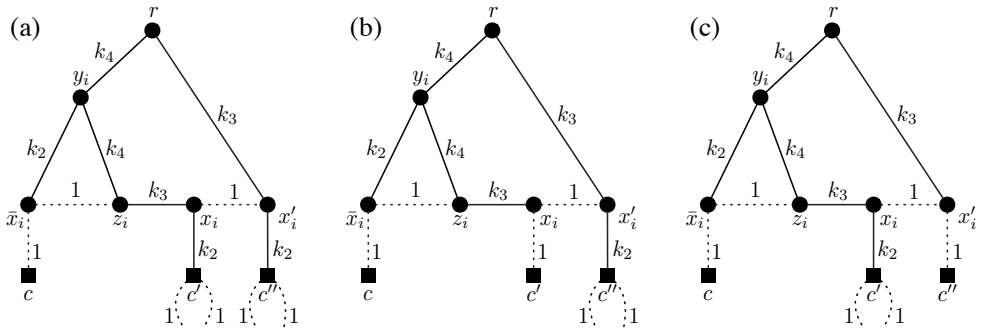


Figure 4.5: The traversal of the x_i-gadget by T when $x_i = 1$. By c, c', and c'' we denote the clauses that contain literals $\bar{x}_i$, x_i and x'_i, respectively. (a) x_i and x'_i are chosen by clauses c' and c''. (b) x'_i is chosen by clause c''. (c) x_i is chosen by clause c'.

Claim 12. *Each x_i-gadget satisfies the following property: for each literal vertex $\tilde{x}_i$, if some edge e of T (not necessarily in the x_i-gadget) is on the r-to-$\tilde{x}_i$ path in T, then $\partial_{G,T}(e)$ contains at least two distinct edges from this gadget other than (y_i, z_i).*

This claim is straightforward: it follows directly from the fact that there are two edge-disjoint paths from r to any literal vertex $\tilde{x}_i \in \{\bar{x}_i, x_i, x'_i\}$ that do not use edge (y_i, z_i).

Claim 13. *For each two-literal clause c, edge (r, c) is not in T.*

For each literal $\tilde{x}_i$ of clause c, there is an r-to-c path via the x_i-gadget, so, together with edge (r, c), G has three disjoint r-to-c paths. Thus, if (r, c) were in T, its congestion would be at least $k_1 + 2 > K$, proving Claim 13.

Claim 14. *All clause vertices are leaves in T.*

To prove Claim 14, suppose there is a clause c that is not a leaf. Then, by Claim 13, c has at least two clause-to-literal edges in T, say $(c, \tilde{x}_i)$ and $(c, \tilde{x}_j)$. We can assume that the last edge on the r-to-c path in T is $e = (c, \tilde{x}_i)$. Clearly, $r \in T_{\tilde{x}_i, c}$ and $\tilde{x}_j \in T_{c, \tilde{x}_i}$. By

Claim 12, at least two edges of the x_j-gadget are in $\partial_{G,T}(e)$, and they contribute at least 2 to $\text{cng}_{G,T}(e)$. We now have some cases to consider.

If c is a two-literal clause, its root-to-clause edge (r, c) is also in $\partial_{G,T}(e)$, by Claim 13. Thus, $\text{cng}_{G,T}(e) \geq k_2 + 3 > K$ (see Figure 4.6a). So assume now that c is a three-literal clause, and let $\tilde{x}_l \neq \tilde{x}_i, \tilde{x}_j$ be the third literal of c. If T contains $(c, \tilde{x}_l)$, the x_l-gadget would also contribute at least 2 to $\text{cng}_{G,T}(e)$, so $\text{cng}_{G,T}(e) \geq k_2 + 4 > K$ (see Figure 4.6b). Otherwise, $(c, \tilde{x}_l) \notin T$, and $(c, \tilde{x}_l)$ itself contributes 1 to $\text{cng}_{G,T}(e)$, so $\text{cng}_{G,T}(e) \geq k_2 + 3 > K$ (see Figure 4.6c).

We have shown that if a clause vertex c is not a leaf in T, then in all cases the congestion of T would exceed K, completing the proof of Claim 14.

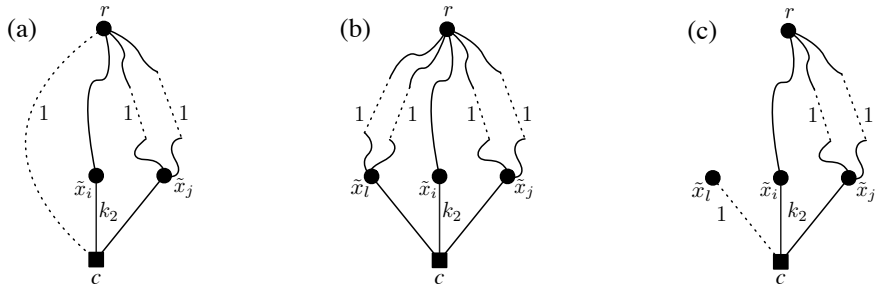


Figure 4.6: Illustration of the proof of Claim 14. In (a), c is a two-literal clause; in (b) and (c), c is a three-literal clause.

Claim 15. *For each x_i-gadget, edge (r, x_i') is in T.*

Towards contradiction, suppose that (r, x_i') is not in T. Let (x_i', c) be the clause-to-literal edge of x_i'. If only one of the two edges $(x_i', x_i), (x_i', c)$ is in T, making x_i' a leaf, then the congestion of that edge is $k_3 + k_2 + 1 > K$. Otherwise, both $(x_i', x_i), (x_i', c)$ are in

T. Because c is a leaf in T by Claim 14, $e = (x_i, x_i')$ is the last edge on the r-to-x_i' path in T. As shown in Figure 4.7a, $\text{cng}_{G,T}(e) \geq k_3 + k_2 + 2 > K$. This proves Claim 15.

Claim 16. *For each x_i-gadget, edge (r, y_i) is in T.*

To prove this claim, suppose (r, y_i) is not in T. We consider the congestion of the first edge e on the r-to-y_i path in T. By Claims 14 and 15, we have $e = (r, x_i')$, all vertices of the x_i-gadget have to be in $T_{x_i', r}$, and $T_{x_i', r}$ does not contain literal vertices of another variable $x_j \neq x_i$. For each literal $\tilde{x}_i$ of x_i, if a clause-to-literal edge $(c, \tilde{x}_i)$ is in T, then the two other edges of c contribute 2 to $\text{cng}_{G,T}(e)$, otherwise $(c, \tilde{x}_i)$ contributes 1 to $\text{cng}_{G,T}(e)$. Then, $\text{cng}_{G,T}(e) \geq k_4 + k_3 + 3 > K$ (see Figure 4.7b), proving Claim 16.

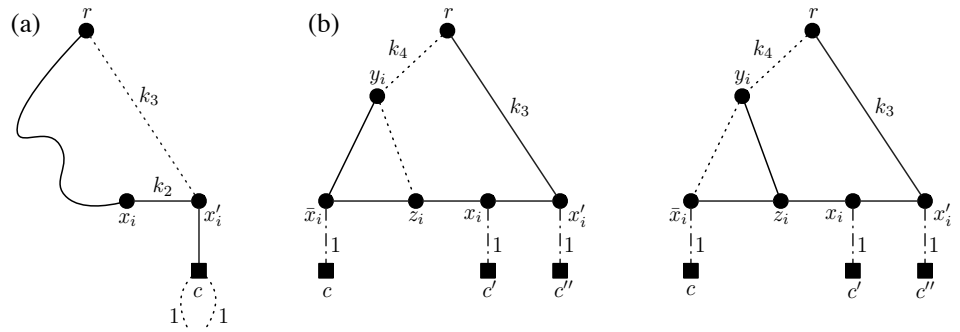


Figure 4.7: (a) Illustration of the proof of Claim 15. (b) Illustration of the proof of Claim 16. Dot-dashed lines are edges that may or may not be in T.

Claim 17. *For each x_i-gadget, exactly one of edges (z_i, x_i) and (x_i, x_i') is in T.*

By Claims 15 and 16, edges (r, y_i) and (r, x_i') are in T. Since the clause neighbor c' of x_i is a leaf of T, by Claim 14, if none of (z_i, x_i), (x_i, x_i') were in T, x_i would not be reachable from r in T. Thus, at least one of them is in T. Now, assume both (z_i, x_i) and (x_i, x_i') are in T (see Figure 4.8a). Then, edge (y_i, z_i) is not in T, as otherwise we would

create a cycle. Let us consider the congestion of edge $e = (r, x'_i)$. Clearly, x_i and x'_i are in $T_{x'_i,r}$. The edges of the two clause neighbors c' and c'' of x_i and x'_i contribute at least 2 to $\text{cng}_{G,T}(e)$, by Claim 14. In addition, by Claim 12, besides e and (y_i, z_i), $\partial_{G,T}(e)$ contains another edge of the x_i-gadget which contributes at least another 1 to $\text{cng}_{G,T}(e)$. Thus, $\text{cng}_{G,T}(e) \geq k_4 + k_3 + 3 > K$ — a contradiction. This proves Claim 17.

Claim 18. *For each x_i-gadget, edge (y_i, z_i) is in T.*

By Claims 15 and 16, the two edges (r, x'_i) and (r, y_i) are in T. Now assume, towards contradiction, that (y_i, z_i) is not in T (see Figure 4.8b). By Claim 17, only one of (z_i, x_i) and (x_i, x'_i) is in T. Furthermore, the clause neighbor c' of x_i is a leaf of T, by Claim 14. As a result, (z_i, x_i) cannot be on the y_i-to-z_i path in T. To reach z_i from y_i, the two edges $(y_i, \bar{x}_i), (\bar{x}_i, z_i)$ have to be in T. Let us consider the congestion of $e = (y_i, \bar{x}_i)$. The edges of the clause neighbor c of $\bar{x}_i$ contribute at least 1 to the congestion of e, by Claim 14. Also, by Claim 12, besides e and (y_i, z_i), $\partial_{G,T}(e)$ contains another edge of the x_i-gadget which contributes at least 1 to $\text{cng}_{G,T}(e)$. In total, $\text{cng}_{G,T}(e) \geq k_4 + k_2 + 2 > K$, reaching a contradiction and completing the proof of Claim 18.

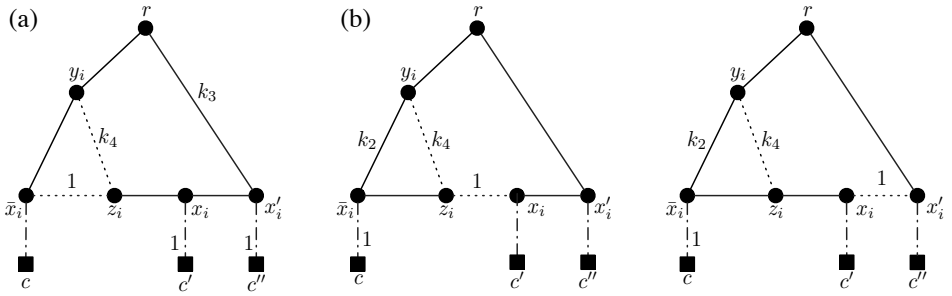


Figure 4.8: (a) Illustration of the proof of Claim 17. (b) Illustration of the proof of Claim 18.

Claim 19. *For each x_i-gadget, if its clause-to-literal edge $(\bar{x}_i, c)$ is in T, then its other two clause-to-literal edges (x_i, c') and (x'_i, c'') are not in T.*

Assume the clause-to-literal edge $(\bar{x}_i, c)$ of the x_i-gadget is in T. By Claim 18, edge (y_i, z_i) is in T. If $(y_i, \bar{x}_i)$ is also in T, edge $(\bar{x}_i, z_i)$ cannot be in T, and it contributes 1 to $\text{cng}_{G,T}(y_i, \bar{x}_i)$. As shown in Figure 4.9a, $\text{cng}_{G,T}(y_i, \bar{x}_i) = k_2 + 3 > K$. Thus, $(y_i, \bar{x}_i)$ cannot be in T. Since c is a leaf of T, edge $(\bar{x}_i, z_i)$ has to be in T, for otherwise $\bar{x}_i$ would not be reachable from r. By Claim 17, one of edges (z_i, x_i) and (x_i, x'_i) is in T. If (z_i, x_i) is in T (see Figure 4.9b), $\text{cng}_{G,T}(y_i, z_i) \geq k_4 + 5 > K$. Hence, (z_i, x_i) is not in T, which implies that (x_i, x'_i) is in T.

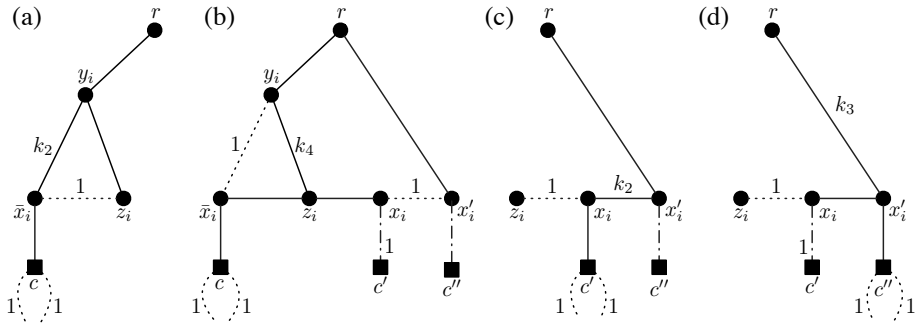


Figure 4.9: Illustration of the proof of Claim 19.

Now, we proceed by contradiction assuming that at least one other clause-to-literal edge of the x_i-gadget is in T. If edge (x_i, c') is in T, $\text{cng}_{G,T}(x_i, x'_i) \geq k_2 + 3 > K$, as shown in Figure 4.9c. Similarly, if (x'_i, c'') is in T, $\text{cng}_{G,T}(r, x'_i) \geq k_3 + 4 > K$ (see Figure 4.9d). So we reach a contradiction in both cases, thus proving Claim 19.

We are now ready to complete the proof of the ($\Leftarrow$) implication in the equivalence ($*$). We use our spanning tree T of congestion at most K to create a truth assignment for

ϕ by setting $x_i = 0$ if the clause-to-literal edge of $\bar{x}_i$ is in T, otherwise $x_i = 1$. By Claim 19, this truth assignment is well-defined. Each clause has one clause-to-literal edge in T which ensures that all clauses are indeed satisfied.

Lower Bound on the Approximation Ratio for STC The proof of Theorem11 immediately improves the lower bound on the approximation ratio for STC:

Corollary 20. *For $c < 1.2$ there is no polynomial-time c-approximation algorithm for STC, unless $\mathbb{P} = \mathbb{NP}$.*

We remark that this hardness result remains valid even if an additive constant is allowed in the approximation bound. This follows by an argument in [10]. (In essence, the reason is that assigning a positive integer weight β to each edge increases its congestion by a factor β.)

4.3 $\mathbb{NP}$-completeness proof of K-STC for bipartite graphs of radius 2 and $K \geq 6$

In this section we establish the following result:

Theorem 21. *For any fixed integer $K \geq 6$, K-STC is $\mathbb{NP}$-complete for bipartite graphs of radius 2, even if they have only one vertex of degree greater than $\max(6, K-2)$.*

First, we introduce a restricted variant of the satisfiability problem, which we name (M2P1N)-SAT, that is used in the reduction. An instance of (M2P1N)-SAT is a boolean expression in conjunctive normal form with the following properties:

- Each clause either contains three positive literals (a 3P-clause), or two positive literals (a 2P-clause), or two negative literals (a 2N-clause). Also, literals in the same clause are of different variables.
- Each variable appears exactly three times: once in a 3P-clause, once in a 2P-clause and once in a 2N-clause.
- Two clauses share at most one variable.

Lemma 22. *(M2P1N)-SAT is* $\mathbb{NP}$*-complete.*

Proof. It is clear that (M2P1N)-SAT belongs to $\mathbb{NP}$. To demonstrate $\mathbb{NP}$-completeness, we show a polynomial-time reduction from the $\mathbb{NP}$-complete problem called BALANCED-3SAT [25]. BALANCED-3SAT is a restriction of the satisfiability problem to boolean expressions in conjunctive normal form where, for each variable x, the positive literal x appears the same number of times as the negative literal $\bar{x}$. We can further assume that every variable appears at least four times, and that, for each clause, all variables that appear in this clause are different.

Given an instance ψ of BALANCED-3SAT, we construct an instance ϕ of (M2P1N)-SAT as follows:

- For each variable x in ψ, if x appears $2t$ times (for some integer $t \geq 2$), create $2t$ new variables $x_0, x_1, \ldots, x_{2t-1}$.
- Replace the t positive occurrences of x by even-indexed variables $x_0, x_2, \ldots, x_{2t-2}$, and replace its t negative occurrences by odd-indexed variables $x_1, x_3, \ldots, x_{2t-1}$.

- Add t clauses of the form $(x_i \vee x_{i+1})$ for $i = 0, 2, \ldots, 2t-2$, and t clauses of the form $(\bar{x}_i \vee \bar{x}_{(i+1) \bmod 2t})$ for $i = 1, 3, \ldots, 2t-1$.

By the construction, ϕ is a correct instance of (M2P1N)-SAT. For each variable x of ψ, its corresponding "cycle" of the newly added two-literal clauses in ϕ ensures that $x_0 = \bar{x}_1 = x_2 = \bar{x}_3 = \ldots = x_{2t-2} = \bar{x}_{2t-1}$. Thus, a truth assignment that satisfies ψ can be converted into a truth assignment that satisfies ϕ by setting the even-indexed variables to the truth value of the original variable in ψ, and the odd-indexed variables to the opposite value. Conversely, a truth assignment that satisfies ϕ can be converted into a truth assignment that satisfies ψ by reversing this process. This shows that ψ is satisfiable if and only if ϕ is satisfiable, completing the proof of the lemma. □

In order to prove Theorem 21, we show a polynomial-time reduction from (M2P1N)-SAT. Given an instance ϕ of (M2P1N)-SAT, we construct a graph G such that

$(*)$ ϕ has a satisfying truth assignment if and only if $\mathrm{stc}(G) \leq K$.

Graph G will be bipartite, of radius 2, and will have only one vertex of degree larger than $\max(6, K-2)$. We will describe G using some double-weighted edges, that we refer to as *fat edges*. As previously discussed in Section 4.1, here we need a specific realization of these double weighted edges, in which weights are realized using the spintop graph. (See Figures 4.1 and 4.2.) For $i \in \{1, 2, 3, 4, 5\}$, let $k_i = K - i$. We start with an empty graph G and proceed as follows:

- Add a *root vertex* r.
- For each variable x of ϕ, add a *variable vertex* x and a *root-to-vertex* edge (r, x).

- For each clause c, add a clause vertex c, and add edges from c to the vertices representing variables whose literals (positive or negative) appear in c. If clause c contains all positive literals, we call its clause-to-variable edges *positive-clause edges*, otherwise its clause-to-variable edges are *negative-clause edges*.

- For each 2P-clause vertex c, add a fat edge (r, c) of double weight $k_5 : k_1$.

- For each 2N-clause vertex c, add a fat edge (r, c) of double weight $k_4 : k_1$.

See Figure 4.10a for an example of a partial graph constructed using the above rules. By routine inspection, taking into account that the weighted edges use the spintop realization, G is bipartite, all vertices are at distance at most 2 from r, and r is the only vertex of degree larger than $\max(6, K - 2)$. We now proceed to show that G satisfies property $(*)$.

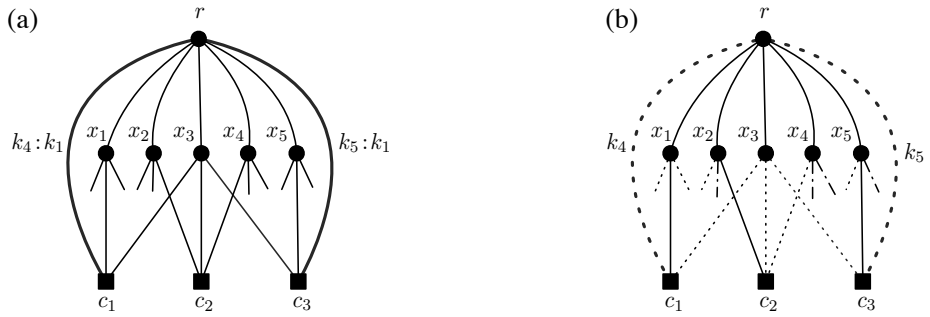


Figure 4.10: (a) An example of a partial graph G for $\phi = c_1 \wedge c_2 \wedge c_3 \wedge \cdots$ where $c_1 = (\bar{x}_1 \vee \bar{x}_3), c_2 = (x_2 \vee x_3 \vee x_4), c_3 = (x_3 \vee x_5)$. Bold lines represent fat edges with given double weights. (b) An example of a partial tree T of G where x_1 is chosen by c_1, x_2 by c_2, x_5 is by c_3. Solid lines are tree edges, dotted lines are non-tree edges, and dot-dashed lines are edges that may or may not be in T. Non-tree double-weighted edges contribute the indicated weights to edge congestion.

($\Rightarrow$) Assume that ϕ has a satisfying truth assignment. From this assignment we construct a spanning tree T of G by adding all root-to-vertex edges, and, for each clause c,

adding to T an edge from c to any variable vertex whose literal satisfies c (see Figure 4.10b). By the construction, T is a spanning tree of G. Note that all clause vertices in T are leaves and all fat edges are non-tree edges.

Now, we proceed to verify that all tree edges of T have congestion at most K. We start with leaf edges of T. The congestion of the leaf edge of a 3P-clause is 3. For a 2P-clause, the congestion of its leaf edge is $K-3$, because its fat edge contributes $k_5 = K-5$. For a 2N-clause, the congestion of its leaf edge is $K-2$, because its fat edge contributes $k_4 = K-4$.

Next, consider the root-to-vertex edge of a variable x_i. If x_i is not chosen to satisfy any clauses, then $\mathrm{cng}_{G,T}(r, x_i) = 4$ (see Figure 4.11a). If it is chosen to satisfy only its 3P-clause, then $\mathrm{cng}_{G,T}(r, x_i) = 5$ (see Figure 4.11b). If it is chosen to satisfy only its 2P-clause, then $\mathrm{cng}_{G,T}(r, x_i) = k_5 + 4 = K-1$ (see Figure 4.11c). If it is chosen to satisfy both its 3P-clause and its 2P-clause, then $\mathrm{cng}_{G,T}(r, x_i) = k_5 + 5 = K$ (see Figure 4.11d). Finally, if it is chosen to satisfy its 2N-clause, then $\mathrm{cng}_{G,T}(r, x_i) = k_4 + 4 = K$ (see Figure 4.11e).

There are also edges inside the realizations of fat edges, but their congestion does not exceed K, by Lemma 10. We have thus shown that the congestions of all edges in T are at most K; that is, $\mathrm{stc}(G) \leq K$.

($\Leftarrow$) Assume T is a spanning tree of G with $\mathrm{cng}_G(T) \leq K$. From T, we will construct a satisfying truth assignment for ϕ. The argument here, while much shorter, has a subtle aspect that was not present in the proof in Section 4.2, namely now it is not necessarily true that all clause vertices in T are leaves. (It's not hard to see that for large K a single branch out of r may visit multiple variables via their 3P-clause vertices.)

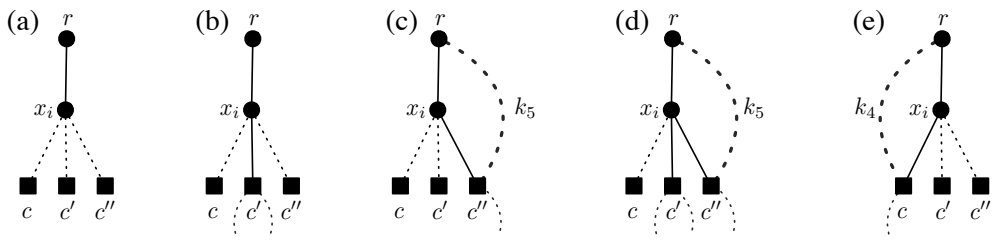


Figure 4.11: By c, c', c'', we denote the 2N-clause, 3P-clause and 2P-clause of x_i respectively. In (a), x_i is not chosen by any clause, it is chosen by c' in (b), by c'' in (c), by both c' and c'' in (d), and by c in (e).

We present two claims showing that T must have a special form that will allow us to define the truth assignment for ϕ.

Claim 23. *For each two-literal clause c, its fat edge (r, c) is not in T.*

For each literal of c, there is an r-to-c path via the variable vertex of this literal. So, together with edge (r, c), G has three disjoint r-to-c paths. Thus, if (r, c) were in T, its congestion would be at least $k_1 + 2 > K$, proving Claim 23.

Claim 24. *For each variable vertex x_i, if its negative-clause edge is in T then its two positive-clause edges are not in T.*

Denote by c, c', c'' the 2N, 3P, 2P-clause vertices of x_i respectively. Since c, c', c'' all contain variable x_i, they cannot share any other variables (by the definition of (M2P1N)-SAT). Therefore, the four literals in c, c', c'' other than x_i and $\bar{x}_i$ must all involve different variables.

Toward contradiction, suppose (x_i, c) and at least one of $(x_i, c'), (x_i, c'')$ are in T. We will estimate the congestion of the first edge $e = (r, v)$ on the r-to-c path in T.

By Claim 23, fat edge (r, c) contributes k_4 to $\text{cng}_{G,T}(e)$. The rest of the argument is based on the following two observations: (i) If a clause $\tilde{c} \in \{c, c', c''\}$ is in $T_{v,r}$, and some variable x is in $\tilde{c}$, then either (r, x) or $(x, \tilde{c})$ is in $\partial_{G,T}(e)$; that is, this x contributes 1 to $\text{cng}_{G,T}(e)$. (This is true whether or not $v = x$. And if $x = x_i$ and $\tilde{c} = c$, then (r, x_i) is the edge that contributes to $\text{cng}_{G,T}(e)$.) On the other hand, (ii) if a clause $\tilde{c} \in \{c', c''\}$ is not in $T_{v,r}$, then $(x_i, \tilde{c})$ contributes 1 to $\text{cng}_{G,T}(e)$.

Now we have some cases to consider. First, if $c' \in T_{v,r}$ and $c'' \notin T_{v,r}$, by the above observations, four different variables in c, c' contribute 4 to $\text{cng}_{G,T}(e)$ and (x_i, c'') contributes 1. In total, $\text{cng}_{G,T}(e) \geq k_4 + 4 + 1 > K$. On the other hand, when $c'' \in T_{v,r}$ and $c' \notin T_{v,r}$, the three different variables of c, c'' contribute 3 while (x_i, c') contributes 1 to $\text{cng}_{G,T}(e)$. Also, the fat edge (r, c'') contributes k_5, by Claim 23. Thus, $\text{cng}_{G,T}(e) \geq k_4 + k_5 + 3 + 1 > K$. Lastly, when both c', c'' are in $T_{v,r}$, the five different variables of c, c', c'' contribute to $\text{cng}_{G,T}(e)$, so $\text{cng}_{G,T}(e) \geq k_4 + 5 > K$. We have thus shown that the congestion of e exceeds K in all cases, completing the proof of Claim 24.

We are now ready to describe the truth assignment for ϕ using T. For each variable x_i, assign $x_i = 0$ if its negative clause edge is in T, otherwise, $x_i = 1$. By Claim 24, the truth assignment is well-defined. By Claim 23, each clause vertex has at least one edge to a variable vertex, which ensures all clauses are satisfied. This completes the proof of Theorem 21.

4.4 Complexity results of K-STC2

In this section, we consider problem K-STCD where, given a graph G, the objective is to determine if G has a depth-D spanning tree of congestion at most K. Here, as before, K is a fixed positive integer. We present the following results:

Theorem 25. *For any fixed integer $K \geq 6$, K-STC2 is $\mathbb{NP}$-complete for bipartite graphs, even if they have only one vertex of degree greater than $\max(6, K-2)$.*

Theorem 26. *For any fixed integer $K \leq 5$, K-STC2 is polynomial-time solvable for bipartite graphs.*

We remark that the complexity status of K-STC2 is independent of whether the root of the spanning tree is specified or not, because there are at most n choices for r. This establish the equivalence of these two versions (with or without the root specified) in terms of polynomial-time solvability or $\mathbb{NP}$-hardness.

4.4.1 $\mathbb{NP}$-completeness proof of K-STC2 for $K \geq 6$

The proof of Theorem 25 can be easily derived from the proof of Theorem 21 in Section 4.3. The reduction remains unchanged. In that construction, the bipartite partition of G has two parts: X, which includes vertices adjacent to the root r (the variable vertices and parts of the spintop gadgets), and $C \cup \{r\}$, which includes the remaining vertices (the clause vertices, the root, and the vertices not adjacent to r in the spintop gadgets). The proof for the forward direction is also identical, since the depth of the spanning tree generated from the proposed construction is already two.

For the reverse implication, suppose T is the depth-two spanning tree with congestion at most K. We present a simple claim about the structure of T:

Claim 27. *All edges incident to r are in T, and all vertices in C are leaves of T.*

Since G does not have any eccentricity-one vertex and the only vertex in G of eccentricity two is r, T has to be rooted at r, which implies that the paths from r to other vertices in T have length at most 2. If an edge $(r, x) \in G$ were not in T, the r-to-x path in T would have length at least 3, which is a contradiction. Thus, T traverses all edges of r. The second part of the claim follows directly from the first part.

In addition to Claim 27, T also has the two properties described in Claim 23 (which can be established using the same argument) and Claim 24 (its proof can be made simpler by considering the fact about clause vertices being leaves of T).

Finally, the truth assignment for ϕ can be created the same way as in Section 4.3.

4.4.2 An algorithm for K-**STC**2 in bipartite graphs for $K \leq 5$

We now prove Theorem 26. We only give an explicit algorithm for $K = 5$. This is because K-STC2 is trivial for $K = 1$, and for $K = 2$, the problem can be solved by a straighforward adaptation of the algorithm in [44], even for general graphs. The cases when $K = 3, 4$ can be handled by slightly modifying (in fact, simplifying) the algorithm for $K = 5$ below. (Alternatively, for $K = 3$, one can adapt the algorithm from [44].)

So let's assume that $K = 5$ and let G be a given bipartite graph. If $\text{rad}(G) > 2$, then G does not have any spanning tree of depth two. If $\text{rad}(G) = 1$, then G must be a complete bipartite graph where one partition contains only one vertex, that is G itself is a

tree of depth one and its congestion is one. Thus, we can assume $\mathrm{rad}(G) = 2$, which means that any depth-two spanning tree of G has to be rooted at a vertex with eccentricity two. There are at most n such vertices, and for each we can check, using the procedure described below, whether there is a depth-two spanning tree T rooted at r such that $\mathrm{cng}_G(T) \leq 5$. Therefore from now on we can assume that this r is already given.

Let X and $C \cup \{r\}$ be the two parts of the bipartition of G. Let E_r be the set of edges incident to r, and $E_s = E \setminus E_r$. We can make the following assumptions (that can be implemented in a pre-processing stage):

- We can assume that all vertices in G have degree at least 2, since removing (repeatedly) degree-one vertices does not affect the spanning tree congestion of the graph.

- By Claim 27, each vertex $c \in C$ has to be a leaf in any depth-two spanning tree rooted at r, and the congestion of its leaf edge is equal to c. Thus, we can also assume that $c \leq 5$ for all $c \in C$.

- Similarly, each edge (r, x) must be in a spanning tree of depth two. With the assumptions above, each edge (x, c) from x to $c \in C$ contributes to the congestion of (r, x), either directly, if it's not in the tree, or indirectly, if it's in the tree (as then the other edges from this c contribute, and there is at least one). Therefore, if $x > 5$ for some $x \in X$, we would have $\mathrm{cng}_{G,T}(r, x) > 5$. So we can assume that $x \leq 5$ for all $x \in X$.

Algorithm outline The general idea of the algorithm is to start with a tree T that contains only edges in E_r and gradually add leaf edges for all vertices $c \in C$. This can be naturally interpreted as assigning vertices in C to vertices in X. If $c \in C$ and $x \in N_G(c)$,

then assigning c to x means that edge (c, x) is being added to T. If it is possible to assign all vertices in C to some vertices in X, while ensuring that the congestions of the edges in E_r do not exceed 5, then T will be the desired spanning tree. In the first phase, we will do this assignment one vertex at a time. Call the assignment $c \to x$ *feasible* if it does not cause the current congestion of (r, x) to exceed 5. Such a feasible assignment can be made safely if it either is forced (say, if c can be assigned to only one vertex in X without exceeding the congestion bound), or it can be made without loss of generality (that is, if we can show that if there is any spanning tree with congestion at most 5, then there is also one that makes this specific assignment). To achieve this, we will carefully track the congestion of the edges in E_r throughout the construction. The first phase will end with all yet unassigned vertices in C of degree 3 or 4. Then the only way to complete the assignments is by adding a matching between C and X, and this is done in the second phase.

Phase 1 Initially T contains only the edges from r to X. During the process, besides these edges, T will also contain one edge (c, x) for each $c \in C$ that is already assigned to $x \in N_G(c)$. For this (not yet spanning) tree T, define the congestion of a vertex $x \in X$ in the current stage of T as:

$$\text{cng}(r, x) \;=\; x + \sum_{c \to x} [c - 2] \tag{4.1}$$

where the sum is over all $c \in C$ that are assigned to x. Thus, when a vertex $c \in C$ get assigned to a vertex $x \in N_G(c)$, the congestion of (r, x) increases by $c - 2 \geq 0$. Note that after this assignment, $\text{cng}(e)$ remains unchanged for $e \in E_r \setminus \{(r, x)\}$ and the congestions of (r, x) is non-decreasing.

Assigning degree-2 vertices. For a vertex c of degree 2, let (x, c) be any of its edges, and assign c to x. The congestion of (r, x) remains unchanged.

Assigning degree-5 vertices. For a vertex c of degree 5, if we assign c to a vertex x, the congestion of (r, x) would increase by 3. Therefore, c can only be assigned to x if the congestion of (r, x) is 2 prior to the assignment, which implies that the only edge in E_s that is incident to x is (c, x). Including (c, x) in T would not affect the congestion of (r, x) in subsequent steps, as c is the only vertex in C that can be assigned to x. If there is no x that satisfies the requirement, we terminate and report failure. If there are multiple feasible choices for such x, we can choose any of them. This is valid, because if $x' \in X$ is another candidate, then x' will not be assigned to any vertices in C and the congestion of (r, x') will remain 2.

Assigning pairs of degree-3 vertices to the same vertex. If there are two degree-3 vertices $c_1, c_2 \in C$ that share the same neighbor x, and $N_G(x) = \{r, c_1, c_2\}$, we can assign both c_1 and c_2 to x. The congestion of (r, x) will increase to 5, and it will remain 5 since x cannot be assigned to any other vertices in C. Similar to the previous step, if there is more than one such choice of x, any option is valid.

Phase 2 After the first phase, we denote by C' the set of yet unassigned vertices in C. The vertices in C' have degree either 3 or 4. Unlike the previous phase, assignments for vertices in C' cannot be made independently. We observe that each of these vertices must be assigned to a different vertex in X because assigning two or more of them to the same x would cause the congestion of (r, x) to exceed 5. (This is because after Phase 1, if two vertices in C' share a neighbor in X then they cannot both have degree 3.) Based on this

observation, we can assume that $|X| \geq |C'|$ – if not, we can report that the congestion is larger than 5. Then an assignment of all vertices in C' forms a perfect matching between C' and X, that is, a matching that covers all vertices in C (but not necessarily in X). Our goal now is to find this matching.

Towards this end, we consider a bipartite subgraph G' of G where one partition consists of the vertices of C', the other partition consists of the vertices in X, and an edge between $c \in C$ and $x \in X$ is included in G' iff $x \to c$ is a feasible assignment. We then determine, in polynomial-time [24], whether G' has a perfect matching. This matching will define the assignments for vertices in C', ensuring that after all assignments are made, the resulting T is now a spanning tree with congestion at most 5. If there is no perfect matching, we report failure.

4.5 Polynomial-time solvability of STC2 in bipartite graphs with vertex degree restrictions

Building upon Section 4.4, we continue to explore the variant of STC2, which involves finding a depth-2 spanning tree with minimum congestion in bipartite graphs. We provide two polynomial-time algorithms for cases when vertex degrees are restricted:

Theorem 28. *STC2 can be solved in polynomial time when all vertices in X have degree at most 3.*

Theorem 29. *STC2 can be solved in polynomial time when all degrees in C have the same degree.*

To prove each theorem, given any positive integer K, we provide an algorithm to construct a depth-2 spanning tree T with congestion at most K (if such a tree exists). This implies the polynomial-time solvability of STC2 in these cases. The proofs are given in Sections 4.5.1 and 4.5.2, respectively.

We use the same notation and terminology as in Section 4.4.2, and we adopt, without loss of generality, similar simplifying assumptions. Let G be the given bipartite graph. We can assume that $\text{rad}(G) = 2$ and the root r of the desired spanning tree is given. We use X and $C \cup \{r\}$ to refer to the two partitions of the vertices of G, and E_r to refer to the set of edges incident to r.

Using the results described in Section 4.4.2, we can solve K-STC2 for $K \leq 5$. Thus, we will assume $K \geq 6$. Also, as in Section 4.4.2, we can assume that $2 \leq v \leq K$ for any $v \in C \cup X$ and $r \geq 2$.

Both algorithms start with a tree T that contains only edges in E_r. The goal is adding leaf edges for all vertices in C while ensuring that the congestion of edges in E_r does not exceed K. For a vertex $x \in X$, the congestion of edge (r, x) in T is defined in the same way as in Equation 4.1.

4.5.1 K-STC2 for bipartite graphs with all degrees in X at most 3

We now present the proof of Theorem 28, namely a polynomial-time algorithm for K-STC2 restricted to bipartite graphs G where the degree of the vertices in X is at most 3. The general idea of this algorithm is similar to the 5-STC2 algorithm described in Section 4.4.2. The process consists of two phases: in the first phase we create assignments for vertices in C that are adjacent to degree-2 vertices in X. Then, in the second phase,

the remaining assignments are determined by a perfect matching in an auxiliary graph H constructed in polynomial time from G. If there is no perfect matching in H, we report failure.

The two phases of the algorithm are as follows:

Phase 1: Assigning to degree-2 vertices. For a vertex $x \in X$ with degree 2, we denote $N_G(x) = \{r, c\}$, we assign $c \to x$. This assignment is safe because the congestion of (r, x) is equal to c, which is at most K by assumption. Moreover, this x cannot be assigned to any other vertices in C which implies that $\text{cng}_{G,T}(r, x)$ will remain unchanged.

Phase 2: Assigning to degree-3 vertices. After the first phase, the remaining vertices in X that are available for assignments have degree 3. Let X' be the set of such vertices, and C' be the set of unassigned vertices in C. Unlike in the 5-STC2 algorithm, we cannot directly use a matching from C' to X' to create feasible assignments because it is possible for two vertices in C' to be assigned to the same vertex in X (not allowed in the second phase of 5-STC2 algorithm). However, we can still capture assigning a pair of vertices in C' to the same vertex in X' by matching this pair to themselves. To accomplish this, we reduce the assignment problem from C' to X' to the problem of finding a perfect matching in an auxiliary graph H (not necessarily bipartite).

The vertices of H consist of all vertices in $X' \cup C'$. In addition, if $|X' \cup C'|$ is odd, we also add r to H. For each $c \in C'$, we add an edge (c, x) to H where $x \in N_G(c)$ if $c + 1 \leq K$. This condition ensures that $c \to x$ is a feasible assignment. Furthermore, for each pair of vertices $c_1, c_2 \in C'$ that share the same neighbor $x \in X'$, if $c_1 + c_2 - 1 \leq K$, we add the edge (c_1, c_2) to H. This condition is equivalent to $\text{cng}_{G,T}(r, x) \leq K$ after both

assignments $c_1 \to x$ and $c_2 \to x$ have been made. Finally, we add edges between any pair of vertices in X' and, if r is in H, we add edges from r to all vertices in X'. Figure 4.12a shows an example construction of H.

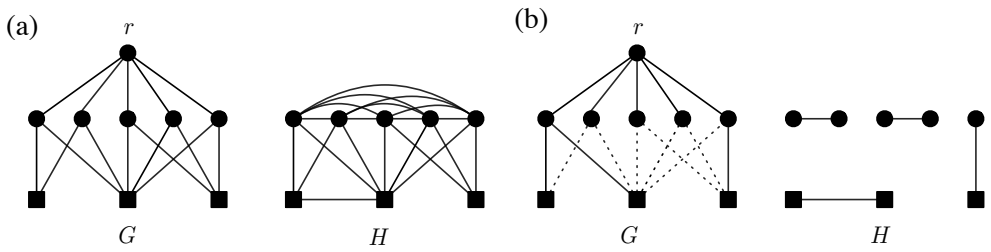


Figure 4.12: (a) An example of H constructed from G for algorithm 6-STC2 in Phase 2. (b) Assignments in G built from a perfect matching in H.

We proceed to find a maximum matching M in H, which can be done in polynomial time [22]. If M is a perfect matching, it will define assignments for all vertices in C' such that these edges combined with the tree T result in a spanning tree of congestion at most K for the graph G. If M is not a perfect matching, we report failure. The following lemma establishes the correctness of this phase:

Lemma 30. *There exist feasible assignments for all vertices in C' if and only if H has a perfect matching.*

Proof. ($\Rightarrow$) Let A denotes the assignments for vertices in C' that represents a depth-2 spanning tree rooted at r with congestion at most K. We will show that H admits a perfect matching M: For each assignment $c \to x$, if x is not assigned to any other vertex in C', we add (c, x) to M; otherwise, x is assigned to exactly one other vertex $c' \in C'$, we add (c, c') to M. The remaining vertices that have not been matched are in X' and r (if it is in H). These vertices can be matched arbitrarily since they form a clique of even size.

($\Leftarrow$) Suppose M is a perfect matching of H. We make assignments for a vertex $c \in C'$ as follows (refer to Figure 4.12b for an example):

- If c is matched with a vertex $x \in X'$ in M, we assign $c \to x$. This assignment is feasible by the construction of H, and we also know that x cannot be assigned to any other vertex according to the condition of the matching.

- If c is matched with another vertex $c' \in C'$, then there exists a vertex $x \in X'$ such that $N_G(x) = \{r, c, c'\}$. In this case, we assign both c, c' to x. By the construction of H, both assignments are feasible, and x is also not used for assignment to any other vertex in C'.

This assignment represents a depth-2 spanning tree rooted at r with congestion at most K. □

4.5.2 K-**STC**2 for bipartite graphs with all degrees in C equal

We now describe a polynomial time algorithm for K-STC2 restricted to bipartite graph G when all vertices in C have degree α, for some positive integer α. This will prove Theorem 29. We can assume that $\alpha \leq K$, for otherwise the congestion of the leaf edges will exceed K. As before, we focus on finding feasible assignments that map each vertex in C to its neighbor in X. These assignments represent a spanning tree rooted at r whose congestion of all edges in E_r do not exceed$K$.

We first consider the case when $\alpha = 2$. For each $c \in C$, if $N_G(c) = \{x_1, x_2\}$, we can assign c arbitrarily to either x_1 or x_2, because the congestions of both (r, x_1) and (r, x_2) are not affected by either assignment.

From now on, we assume that $\alpha \geq 3$. The idea of the algorithm is to express the assignments for vertices in C via the maximum $s-t$ flow in an auxiliary flow network F that can be constructed in polynomial time from G. The graph F includes all edges and vertices of G. All the edges are directed from r to X and from X to C. Additionally, F has a source vertex s and directed edge (s, r), and a sink vertex t with directed edges from vertices C to t. We use $c(u, v)$ to denote the capacity of the edge (u, v). The capacities of all edges in F are defined as follows:

- $c(s, r) = |C|$
- For each vertex $x \in X$, $c(r, x) = \lfloor \frac{K-x}{\alpha-2} \rfloor$
- For each edge (x, c) in F where $x \in X$ and $c \in C$, $c(x, c) = 1$
- For each vertex $c \in C$, $c(c, t) = 1$

We then find, in polynomial time, a maximum $s-t$ flow f in F. As we will show, if f has an $s-t$ flow of value $|C|$, this flow will define a feasible assignment for all vertices in C representing a depth-2 spanning tree rooted at r with congestion at most K. If the maximum flow value is less than $|C|$, we report failure. The following lemma establishes the correctness of the reduction:

Lemma 31. *There exists feasible assignments for all vertices in C if and only if F has an $s-t$ flow of value $|C|$.*

Proof. ($\Rightarrow$) Suppose F has an $s-t$ flow f of value $|C|$. We denote by $f(u, v)$ the flow value on the edge (u, v). Since $|C|$ is integral and all capacities are integral, we can assume that

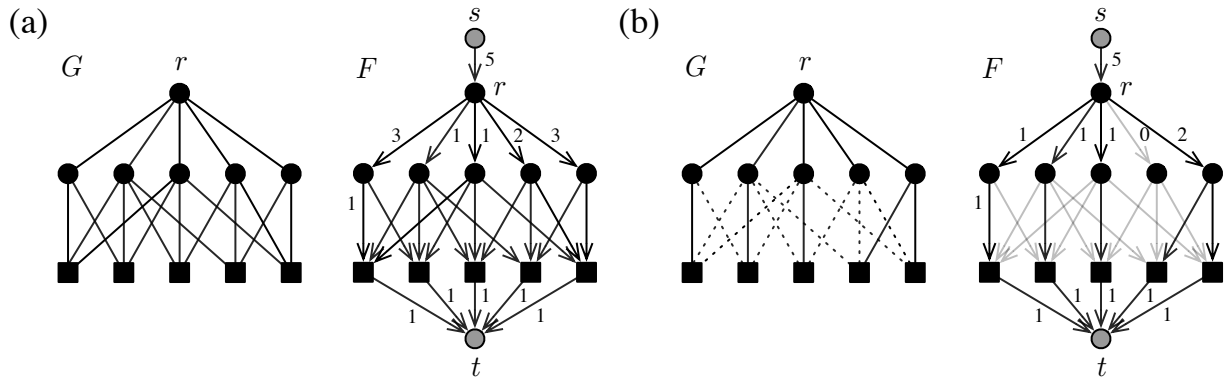


Figure 4.13: (a) An example of the auxiliary network F (on the right) constructed from G (on the left). Edges from X to C have capacity 1, all other edges have capacities as shown. (b) On the left, a maximum flow in F. Dark edges have flows with shown values and light edges have no flow. On the right, the assignment obtained from this flow.

flow values of f on all edges are integral. Therefore, for each vertex $c \in C$, $f(c,t) = 1$, which implies that there is exactly one vertex $x \in X$ with $f(c,x) = 1$. We then assign $c \to x$.

Next, we need to verify that in the corresponding tree, the congestions of the edges in E_r are at most K. For each vertex $x \in X$, the number of vertices in C that can be assigned to this x is bounded by $c(r,x)$. By Equation 4.1, $\text{cng}(r,x) \leq x + c(r,x)\,(\alpha - 2) \leq K$, which completes the proof of this implication.

($\Leftarrow$) Suppose there exists a feasible assignment for all vertices in C. From this assignment we will construct an $s-t$ flow f for F with value $|C|$. For each vertex $c \in C$, if c is assigned to $x \in X$, then we define $f(x,c) = 1$ and $f(x',c) = 0$ for all $x' \in N_G(c) \setminus \{x\}$. Next, for each vertex $x \in X$, we define $f(r,x) = n_x$ where n_x is the number of vertices in C that are assigned to x. Due to the congestion bound on (r,x), we have $n_x \leq \frac{K-x}{\alpha-2}$. However, since n_x is integral, we have $n_x \leq \lfloor \frac{K-x}{\alpha-2} \rfloor = c(r,x)$. Lastly, we let $f(c,t) = 1$ for

each $c \in C$ and $f(r, s) = |C|$. Clearly, the constructed flow f has value $|C|$, and it satisfies flow conservation and capacity constraints of F. $\square$

Bibliography

[1] Alison Templeton Adams. *Flow analysis of distribution systems containing elevated reservoirs by the Hardy Cross method.* PhD thesis, Georgia Institute of Technology, 1955.

[2] T. Altman and P. F. Boulos. Convergence of Newton method in nonlinear network analysis. *Math. Comput. Model.*, 21(4):35–41, February 1995.

[3] Fernando Alvarruiz Bermejo, Fernando Martínez Alzamora, and Antonio Manuel Vidal Maciá. Improving the efficiency of the loop method for the simulation of water distribution networks. *Journal of Water Resources Planning and Management*, 141(10):1–10, 2015.

[4] Nobuaki Aoki, Ryota Umei, Atsufumi Yoshida, and Kazuhiro Mae. Design method for micromixers considering influence of channel confluence and bend on diffusion length. *Chemical Engineering Journal*, 167(2-3):643–650, 2011.

[5] Giuseppe Di Battista and Roberto Tamassia. Algorithms for plane representations of acyclic digraphs. *Theoretical Computer Science*, 61(2):175 – 198, 1988.

[6] András A Benczúr and David R Karger. Approximating $s-t$ minimum cuts in $\tilde{O}(n^2)$ time. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 47–55, 1996.

[7] Sandeep Bhatt, Fan Chung, Tom Leighton, and Arnold Rosenberg. Optimal simulations of tree machines. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86, page 274–282, USA, 1986. IEEE Computer Society.

[8] Sukanta Bhattacharjee, Bhargab B. Bhattacharya, and Krishnendu Chakrabarty. *Algorithms for Sample Preparation with Microfluidic Lab-on-Chip.* River Publishers Series in Biomedical Engineering. River Publishers, 2019.

[9] Sukanta Bhattacharjee, Sudip Poddar, Sudip Roy, Juinn-Dar Huang, and Bhargab B. Bhattacharya. Dilution and mixing algorithms for flow-based microfluidic biochips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(4):614–627, April 2017.

[10] Hans Bodlaender, Fedor Fomin, Petr Golovach, Yota Otachi, and Erik Leeuwen. Parameterized complexity of the spanning tree congestion problem. *Algorithmica*, 64:1–27, 09 2012.

[11] Hans L. Bodlaender, Kyohei Kozawa, Takayoshi Matsushima, and Yota Otachi. Spanning tree congestion of k-outerplanar graphs. *Discrete Mathematics*, 311(12):1040–1045, 2011.

[12] Paul F. Boulos, Kevin E. Lansey, and Bryan W. Karney. *Comprehensive water distribution systems analysis handbook for engineers and planners*. MWH Soft, 2006.

[13] Dejan Brkić. An improvement of Hardy Cross method applied on looped spatial natural gas distribution networks. *Applied Energy*, 86:1290–1300, 07 2009.

[14] Leizhen Cai and Derek G. Corneil. Tree spanners. *SIAM Journal on Discrete Mathematics*, 8(3):359–387, 1995.

[15] COMSOL, Inc. Comsol.

[16] Hardy Cross. Analysis of flow in networks of conduits or conductors. *Engineering Experiment Station*, Bulletin No 286, 1936.

[17] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

[18] Feodor F. Dragan, Fedor V. Fomin, and Petr A. Golovach. Spanners in sparse graphs. *Journal of Computer and System Sciences*, 77(6):1108–1119, 2011.

[19] Yuval Emek and David Peleg. Approximating minimum max-stretch spanning trees on unweighted graphs. *SIAM Journal on Computing*, 38(5):1761–1781, 2009.

[20] Sándor P. Fekete and Jana Kremer. Tree spanners in planar graphs. *Discrete Applied Mathematics*, 108(1):85–103, 2001. Workshop on Graph Theoretic Concepts in Computer Science.

[21] Wai Shing Fung, Ramesh Hariharan, Nicholas JA Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, pages 71–80, 2011.

[22] Zvi Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys (CSUR)*, 18(1):23–38, 1986.

[23] Antonio Gameiro Lopes. Implementation of the Hardy-Cross method for the solution of piping networks. *Computer Applications in Engineering Education*, 12:117 – 125, 01 2004.

[24] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[25] Klemens Hägele, Colm Ó Dúnlaing, and Søren Riis. The complexity of scheduling tv commercials. *Electronic Notes in Theoretical Computer Science*, 40:162–185, 2001. MFCSIT2000, The First Irish Conference on the Mathematical Foundations of Computer Science and Information Technology.

[26] Stephen C. Jacobson, Timothy E. McKnight, and J. Michael Ramsey. Microfluidic devices for electrokinetically driven parallel and serial mixing. *Analytical Chemistry*, 71(20):4455–4459, 1999.

[27] L. Jay. A note on q-order of convergence. *BIT Numerical Mathematics*, 41:422–429, 03 2001.

[28] Weiqing Ji, Tsung-Yi Ho, and Hailong Yao. More effective randomly-designed microfluidics. *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2018.

[29] Kim Jungkyu, Johnson Michael, Hill Parker, and Gale Bruce. Microfluidic sample preparation: cell lysis and nucleic acid purification. *Integrative Biology*, 1:574–586, Oct 2009.

[30] Athina S. Kastania, Katerina Tsougeni, George Papadakis, Electra Gizeli, George Kokkoris, Angeliki Tserepi, and Evangelos Gogolides. Plasma micro-nanotextured polymeric micromixer for dna purification with high efficiency and dynamic range. *Analytica Chimica Acta*, 942:58–67, 2016.

[31] Telikepalli Kavitha, Christian Liebchen, Kurt Mehlhorn, Dimitrios Michail, Romeo Rizzi, Torsten Ueckerdt, and Katharina A. Zweig. Survey: Cycle bases in graphs characterization, algorithms, complexity, and applications. *Comput. Sci. Rev.*, 3(4):199–243, November 2009.

[32] Samir Khuller, Balaji Raghavachari, and Neal Young. Designing multi-commodity flow trees. *Information Processing Letters*, 50(1):49–55, 1994.

[33] Brian J. Kirby. *Micro- and nanoscale fluid mechanics: transport in microfluidic devices.* Cambridge University Press, 2013.

[34] Kyohei Kozawa and Yota Otachi. Spanning tree congestion of rook's graphs. *Discussiones Mathematicae Graph Theory*, 31(4):753–761, 2011.

[35] Kyohei Kozawa, Yota Otachi, and Koichi Yamazaki. On spanning tree congestion of graphs. *Discrete Mathematics*, 309(13):4215–4224, 2009.

[36] Christian Löwenstein. *In the Complement of a Dominating Set.* PhD thesis, Technische Universität at Ilmenau, 2010.

[37] Huong Luu and Marek Chrobak. *Modeling Fluid Mixing in Microfluidic Grids*, pages 149–159.

[38] Huong Luu and Marek Chrobak. Better hardness results for the minimum spanning tree congestion problem. In Chun-Cheng Lin, Bertrand M. T. Lin, and Giuseppe Liotta, editors, *WALCOM: Algorithms and Computation - 17th International Conference and Workshops, WALCOM 2023, Hsinchu, Taiwan, March 22-24, 2023, Proceedings*, volume 13973 of *Lecture Notes in Computer Science*, pages 167–178, Cham, 2023. Springer.

[39] Hans Bruun Nielsen. Methods for analyzing pipe networks. *Journal of Hydraulic Engineering*, 115(2):139–157, 1989.

[40] Yoshio Okamoto, Yota Otachi, Ryuhei Uehara, and Takeaki Uno. Hardness results and an exact exponential algorithm for the spanning tree congestion problem. In *Proceedings of the 8th Annual Conference on Theory and Applications of Models of Computation*, TAMC'11, page 452–462, Berlin, Heidelberg, 2011. Springer-Verlag.

[41] J. M. Ortega. The Newton-Kantorovich theorem. *The American Mathematical Monthly*, 75(6):658–660, 1968.

[42] M.I Ostrovskii. Minimal congestion trees. *Discrete Mathematics*, 285(1):219–226, 2004.

[43] Yota Otachi. *A Survey on Spanning Tree Congestion*, pages 165–172. Springer International Publishing, Cham, 2020.

[44] Yota Otachi, Hans L. Bodlaender, and Erik Jan Van Leeuwen. Complexity results for the spanning tree congestion problem. In *Proceedings of the 36th International Conference on Graph-Theoretic Concepts in Computer Science*, WG'10, page 3–14, Berlin, Heidelberg, 2010. Springer-Verlag.

[45] Brian M. Paegel, William H. Grover, Alison M. Skelley, Richard A. Mathies, and Gerald F. Joyce. Microfluidic serial dilution circuit. *Analytical Chemistry*, 78(21):7522–7527, 2006. PMID: 17073422.

[46] Werner C. Rheinboldt. A unified convergence theory for a class of iterative processes. *SIAM Journal on Numerical Analysis*, 5(1):42–63, 1968.

[47] Romeo Rizzi. Minimum weakly fundamental cycle bases are hard to find. *Algorithmica*, 53(3):402–424, March 2009.

[48] Arnold L. Rosenberg. Graph embeddings 1988: Recent breakthroughs, new directions. In John H. Reif, editor, *VLSI Algorithms and Architectures*, pages 160–169, New York, NY, 1988. Springer New York.

[49] Uri Shamir. Water distribution system analysis. *J. Hyd. Div. ASCE*, 94, 01 1968.

[50] Uri Shamir and Charles Howard. Engineering analysis of water-distribution systems. *Journal American Water Works Association*, 69:510–514, 09 1977.

[51] Manish Kumar Singh and Vassilis Kekatos. On the flow problem in water distribution networks: Uniqueness and solvers. *IEEE Transactions on Control of Network Systems*, 8(1):462–474, 2021.

[52] John J. Sopka. Functional analysis in normed spaces (L. V. Kantorovich and G. P. Akilov). *SIAM Review*, 11(3):412–413, 1969.

[53] Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

[54] Todd M. Squires and Stephen R. Quake. Microfluidics: Fluid physics at the nanoliter scale. *Rev. Mod. Phys.*, 77:977–1026, Oct 2005.

[55] D.J. Stephenson. *Pipeflow Analysis*. ISSN. Elsevier Science, 1984.

[56] Roberto Tamassia and Ioannis G. Tollis. A unified approach to visibility representations of planar graphs. *Discrete & Computational Geometry*, 1(4):321–341, Dec 1986.

[57] Junchao Wang, Philip Brisk, and William H. Grover. Random design of microfluidics. *Lab on a Chip*, 16:4212–4219, 2016.

[58] Don J. Wood. Algorithms for pipe network analysis and their reliability. Technical Report 127, Kentucky Water Resources Research Center, 1981.

[59] Paul Yager, Thayne Edwards, Elain Fu, Kristen Helton, Kjell Nelson, Milton R. Tam, and Bernhard H. Weigl. Microfluidic diagnostic technologies for global public health. *Nature News*, Jul 2006.

[60] Ryo Yoshinaka. Higher-order matching in the linear lambda calculus in the absence of constants is NP-complete. In Jürgen Giesl, editor, *Term Rewriting and Applications*, pages 235–249, 2005.