Solving Graph Flow Problems with Neural Networks: A Lagrangian Duality Approach

Abstract

The problem of computing minimum cost graph flows has important applications in the management of transportation systems. Existing approaches to solving this problem, however, have high computational complexity and cannot scale to larger graphs. This research presents an entirely new approach to solving the minimum cost flow problem when the cost function is convex. The proposed technique uses message-passing graph neural networks to learn graph flows. This work addresses the question: can graph neural networks compute minimum cost flows faster than existing techniques? The goal of learning minimum cost flows fits within the larger context of using graph neural networks to solve algorithmic problems.

Training a neural network to learn minimum cost graph flows is challenging for two reasons. Firstly, the model must output arc flows which constitute a valid circulation. Secondly, ground-truth labels are difficult to compute because existing algorithms are inefficient. These problems are solved by introducing a powerful reparameterization. Instead of directly learning flow values, the neural network learns relative flow proportions. This formulation allows the neural network to always produce valid flows. This maintenance of feasibility gives rise to a novel training technique based on Lagrangian duality. This method, named DualityLoss, uses separate neural networks for the primal and dual optimization problems. The models are then jointly trained with the goal of minimizing the estimated duality gap. Given a strongly dual problem and neural networks which maintain feasibility, this objective encourages the primal and dual models to produce optimal solutions. This training technique uses properties of Lagrangian duality to remove the need for labeled datasets.

This novel approach is used to train a graph neural network on the task of minimum cost graph flows. The graph neural network features an architecture which is more compatible with high-diameter road graphs. This architecture uses ideas of graph attention, gated state updates, and neighborhood expansion. The neural network approach is evaluated against pure optimization techniques on a set of real-world road graphs. These results show that the model learns flows which are close to the optimal cost. Furthermore, the neural network computes these flows significantly faster than the best optimization algorithm. Qualitative examples show how the model learns to coordinate the behavior of vertices within the graph. These results are encouraging because the neural network learns these abilities without ever receiving ground-truth information.
## Contents

1 Introduction 5

2 Related Work 7
   2.1 Graph Neural Network Architectures 7
   2.2 Graph Neural Networks for Algorithmic Graph Problems 8
   2.3 Dueling Neural Networks 9
   2.4 Leveraging Duality in Neural Networks 10
   2.5 Minimum Cost Flows 11

3 Notation and Problem Statement 11
   3.1 Mathematical Notation 12
   3.2 Graph Notation 12
   3.3 The Minimum Cost Flow Problem 13
   3.4 Graph Neural Networks 13

4 Training Neural Networks Using Lagrangian Duality 14
   4.1 Motivating DualityLoss 14
   4.2 Defining DualityLoss for Constrained Optimization Problems 15
   4.3 DualityLoss for Minimum Cost Flows 17
   4.4 Producing Valid Flows with Neural Networks 17
   4.5 Theoretical Properties of the Iterative Flow Solver 19
      4.5.1 Derivation 19
      4.5.2 Properties 20
   4.6 The Minimum Cost Flow Dual Problem 20

5 Graph Neural Network Design 22
   5.1 Motivation and Architecture Overview 22
   5.2 End-to-End Model 23
   5.3 Implementation 24
      5.3.1 Encoder 24
      5.3.2 Graph Attention and Neighborhood Lookahead 24
      5.3.3 Decoder 26
   5.4 Converting Edge Scores into Flow Proportions 26

6 Evaluation 27
   6.1 An Illustration of Computed Graph Flows 28
      6.1.1 Datasets and Training 30
   6.2 Comparing the Neural Network Approach to Baseline Techniques: Flow Cost 30
   6.3 Comparing the Neural Network Approach to Baseline Techniques: Computation Time 32
   6.4 Evaluating the Design Choices of the Graph Neural Network Architecture 33
   6.5 Assessing the Impact of the Iterative Flow Solver 34
   6.6 Details of the Baseline Solvers 34
      6.6.1 Optimization Algorithms 34
1 Introduction

This project presents a graph-neural-network-based approach to the minimum cost graph flow problem.

Graph neural networks [18, 36] have been used to solve both classification and prediction problems on graphs. For example, given a social network graph where each vertex is an account and each edge is a friendship, graph neural networks can learn patterns in the social graph which identify fraudulent accounts. Graph neural networks approach such classification problems by (1) defining an initial state $x_u$ for each vertex $u$, (2) repeatedly updating these states using neighboring nodes i.e. $x_u \leftarrow f(\{x_w\})$ for all neighbors $w$ of $u$, and (3) predicting the final classification labels $y_u \leftarrow g(x_u)$ for all vertices. The functions $f$ and $g$ are implemented as neural networks which are trained to maximize the classification accuracy.

Optimization problems on graphs, however, are often of a different style. Consider the famous maximum flow problem. This problem involves finding the maximum possible flow between a source $s$ and a sink $t$ which both forms a circulation and abides by the given edge capacities. As with many graph optimization problems, the maximum flow problem can be represented within the framework of constrained optimization. The maximum flow problem can be efficiently solved by techniques such as the Ford-Fulkerson method. Not all graph optimization problems, however, can be solved using efficient, specialized algorithms; in many cases, finding solutions requires resorting to general-purpose optimization techniques.

One variation on traditional maximum flow problem is the problem of finding minimum cost flows (MCFs). This problem involves multiple sources and multiple sinks. Each source is assigned an amount of supply, and each sink has a given demand. The goal is to find a flow from sources to sinks which both satisfies all demands and minimizes a cost function. This task has practical applications in both transportation systems and vehicle routing [7]. Despite its close relation to the maximum flow task, the minimum cost flow problem is inefficient to solve when the cost function is nonlinear [20]. This inefficiency even extends to cases where the cost is convex [7, Chapter 9]. As convex cost functions appear in real-world scenarios such as route selection [45], the inefficiencies of existing solvers make them unsuitable for real-time flow computations in practical settings.

This thesis presents a new approach to the MCF problem. This approach uses graph neural networks to learn MCFs when the cost function is nonlinear and convex. Unlike traditional algorithms, neural networks offload a significant amount of computation to the training phase. The information gained during training allows the neural network to efficiently generalize to new problem instances. Furthermore, hardware components, such as GPUs, improve the wall-to-wall runtime of neural network models. For these reasons, graph neural networks can theoretically compute minimum cost flows in near-real-time.

The task of learning MCFs with graph neural networks is difficult for two reasons. Firstly, the desired graph neural network must produce solutions which satisfy the problem’s constraints. To ensure the graph neural network outputs valid flows, the neural network learns relative flow proportions instead of actual flow values. These proportions form probability distributions over the outgoing edges of each vertex. The flow proportions are then used to construct a valid flow. As neural networks can easily output distributions by using operations such as softmax, this method enables graph neural networks to always produce valid flows.
The second challenge is that neural networks require large amounts of labeled training data. For algorithmic tasks, this data is often generated by applying an existing algorithm to randomly generated problem instances. For example, to train a neural network for the task of shortest paths, one can create data labels using Dijkstra's shortest path algorithm. This strategy, however, is not feasible when existing solvers have high computational complexities. To address this problem, a novel training method for constrained optimization problems is introduced. This technique, called DualityLoss, uses neural network models for both the primal and the dual optimization problems. These models are then trained with the joint objective of minimizing the difference between the estimated primal and dual costs. Figure 1 depicts this technique. This objective encourages the models to find the optimal solution as long as both models produce feasible solutions and the duality gap is zero. The DualityLoss method is motivated by the fact that, for problems with no duality gap, optimal solutions occur when primal and dual values coincide. DualityLoss removes the need for labeled data, thus allowing the neural network to be trained on large problem instances for which exact solutions are not feasible to compute.

The proposed graph neural network approach is evaluated on a set of real-world road graphs obtained from Open Street Map [32]. The graph neural network uses an architecture based on ideas from graph attention [42], gated state updates [25], and neighborhood expansion [2]. This design addresses some of the problems related to long-range information retention which are induced by high-diameter road graphs. The graph neural network learns flows which are close to the optimal cost, and it computes these flows far faster than the best pure optimization technique. The graph neural network displays the ability to coordinate the behavior of multiple vertices, and this result is encouraging because the neural network learns this strategy without ever receiving ground-truth labels.

This application of graph neural networks to the MCF problem fits within the larger context of using graph neural networks for algorithmic tasks. While graph neural networks have been applied to problems such as boolean satisfiability [37] and path-finding [25], this thesis presents the first graph neural network approach to solving flow problems.

This report proceeds as follows. Section 2 puts forth a selection of related work. Section 3 presents the notation, formal problem statement, and relevant background information.
Section 4 outlines the DualityLoss technique for training neural networks. Section 5 discusses the graph neural network design. Section 6 presents an evaluation of this technique on real-world road graphs, and these results are reflected upon in section 7. Section 8 discusses ideas for future work, and section 9 concludes the report. Section 10 contains a set of proofs for theorems introduced in earlier sections.

2 Related Work

The neural network method developed in this thesis uses ideas from graph neural networks, reinforcement learning, adversarial networks, and dual optimization problems. This section places the ideas present in this thesis within the context of related work in these areas.

2.1 Graph Neural Network Architectures

Graph neural networks are an extension of recursive neural networks which operate on graph-structured data. Gori et al. [15] and Scarselli et al. [36] proposed the original graph neural network model, GNN. GNN uses a message-passing framework; at each step, every node updates its state by aggregating the states of its neighbors. An output function maps node states to output values. The output and transition functions are implemented as feed-forward neural networks. Training these neural networks using standard backpropagation, however, is memory-inefficient because it requires storing node states for each step of the unrolled network. To address this problem, GNN executes transition steps until the node states reach a fixed point. Given this fixed point, the neural networks can use the Almeida-Pineda algorithm [3, 33] to perform backpropagation without storing the entire history of states. To ensure that the fixed point iteration converges, GNN requires the transition function to be a contraction mapping.

The restriction of the transition model to a contraction mapping limits GNN’s ability to retain information across long distances. The gated graph neural network (GG-NN) design addresses these problems [25]. GG-NNs remove the contraction mapping restriction and instead run a fixed number of transition steps. Gradients are then computed using backpropagation through time. This decision allows for a less constricted model at the cost of more memory. GG-NNs use gated recurrent units (GRUs) [11] to update node states during each transition step. The application of GRUs allows the model to better retain information over long distances. This quality is important when applying graph neural networks to high-diameter graphs. For this reason, the model presented in this work makes use of gated node updates.

While recurrent units help address the issue of long-range information retention, they do not completely solve the problem. N-GCN approaches this problem from a different perspective [2]. This model builds on Kipf and Welling’s graph convolutional network (GCN) [23]. GCN extends the traditional convolution operation to graph-structured data by using the normalized graph Laplacian. N-GCN augments GCN by applying multiple graph convolution operators to aggregate information from different neighborhoods. For each vertex, these neighborhood levels are formed from nodes located at increasing random walk dis-
The models above consider all neighboring states equally when aggregating information. This assumption of equality may not be optimal, and nodes should ideally focus on neighbors which are more relevant to producing the correct output. This idea is at the core of graph attention networks (GATs) [42]. At each transition step, nodes aggregate neighboring states using a weighted average. These weights are formed by applying a softmax operator to the output of a feed-forward neural network. The GAT model represents an extension of sequence attention models [6] to graph-structured domains. The concept of graph attention is compatible with the task of computing graph flows. Flow values can be interpreted as the amount of emphasis placed on a given edge. This interpretation lends itself to the attention model, as attention enables nodes to bias towards neighbors which lead to more optimal solutions. Based on the intuition relating graph attention and graph flows, the architecture in this thesis adopts an attention-based aggregation function.

2.2 Graph Neural Networks for Algorithmic Graph Problems

Graph neural networks have seldom been applied to graph-theoretic problems of high computational complexity. NeuroSAT uses a graph neural network to solve the boolean satisfiability (SAT) problem, and it is the most successful graph neural network approach to an NP-Complete problem [37]. This model is trained as a classifier on a randomly generated set of SAT problems with between 10 and 40 variables and roughly 200 clauses. As these samples are small, existing SAT solvers can efficiently generate data labels. NeuroSAT generalizes to larger SAT problems by executing more message-passing steps at testing time. Despite being a classifier, the model encodes variable assignments in within intermediate node representations.

Prates, Avelar, and Lemos et al. build on this work by developing a graph neural net-
work technique for solving the traveling salesman decision problem (TSP) [34]. The authors train this model on graphs which contain between 20 and 40 nodes. These graphs are small enough to use existing algorithms to compute optimal costs. From each training graph, the authors generate positive and negative samples. These samples have respective costs which are higher and lower than the optimal value. The margin used to create these deviations allows the model to learn the optimal TSP cost within a certain error bound.

Gated graph sequence neural networks (GGS-NNs) are used to address the problem of path-finding [25]. GGS-NN is a graph neural network architecture capable of predicting sequences. The model uses two gated graph neural networks: one to create node states and one to produce outputs. Figure 2 shows an example of this architecture. The authors apply GGS-NNs to the task of general path-finding, shortest paths, and Eulerian circuits. GGS-NN solves these problems by producing sequential outputs until it predicts a special “stop” state. For all tasks, GGS-NN uses training samples with data labels denoting the optimal sequence. Even with limited training examples, GGS-NN achieves 100% accuracy for both shortest paths and Eulerian circuits on small problem instances.

The goal of learning MCFs relates to this previous work in that it also involves using neural networks to solve algorithmic graph problems. The technique presented in this thesis, however, uses a different approach to training. In the related work above, graph neural networks are trained using coarse-grained labels. For tasks such as shortest paths, this strategy is possible due to existing efficient algorithms. For problems where no efficient algorithms exist, training sets are constructed from small instances. In contrast to these approaches, the DualityLoss technique uses no data labels. Instead, it leverages the properties of the primal and dual optimization problems. DualityLoss is useful for the MCF problem because (1) MCFs are inefficient to solve exactly and (2) neural networks can always produce feasible solutions by learning flow proportions. The requirement of feasibility is difficult to satisfy for problems such as TSP.

2.3 Dueling Neural Networks

The term dueling neural networks refers to architectures which train a pair of neural network models against one another. Generate Adversarial Networks (GANs) fall under this class of models [14]. GANs are composed of two neural network models: a generator and a discriminator. The generator model uses seed values to generate outputs. The discriminator model evaluates these outputs and assesses whether they belong to the true dataset. The two models engage in a minimax game in which the generator tries to create realistic samples which fool the discriminator. The generator and discriminator are measured against each other, and training occurs by taking turns updating the generator and the discriminator.

AlphaGo Zero is another example of a dueling neural network architecture [40]. AlphaGo Zero is a deep reinforcement learning system which has achieved world-record performance on the games of Go, Shogi, and Chess. The system couples a deep neural network with a Monte-Carlo tree search algorithm to explore game states. Rather than learning by playing against humans, AlphaGo Zero learns by playing against itself. The win/loss outcome of each game decides the reward propagated to each model instance. This technique of self-play has allowed AlphaGo Zero to adopt strategies which fall outside of the conventional human theory surrounding these games.
The DualityLoss technique relates to these ideas in that it also trains a pair of neural networks based on one another. DualityLoss, however, solves a different type of problem than the ones considered by both GANs and AlphaGo Zero. Namely, DualityLoss is a technique to train neural networks for constrained optimization problems. Furthermore, dueling architectures involve competition between opposing neural networks. The primal and dual models of DualityLoss, however, have the collaborative goal of minimizing the estimated duality gap. This goal comes from directly from Lagrangian duality theory.

2.4 Leveraging Duality in Neural Networks

To the best of our knowledge, the DualityLoss technique is a novel contribution of this work. The concept of duality, however, has been used elsewhere to aid learning algorithms. Shalev-Schwarz and Singer use primal and dual optimization problems to train an online linear classifier [38]. Online classifiers are evaluated by the number of classification mistakes made over time. These mistake values are often relative to the optimal prediction algorithm which has the benefit of observing the entire sequence of examples. Shalev-Schwarz and Singer formulate an online linear classifier as a constrained optimization problem subject to a given relative mistake bound. Rather than solving the primal problem, the authors instead use the Lagrangian dual formulation. This substitution makes sense because the optimal dual and primal values coincide due to strong duality. The authors prove that the dual optimization is non-decreasing, and, if the average increase is sufficiently large, then the dual optimization meets the desired relative mistake bound. The authors later apply this same technique to the more general problem of repeated convex games by instead using Fenchel duality [39].

The duality technique for online linear classification leverages the strong duality of the linear classification optimization problem. This idea is similar to the use of strong duality by the DualityLoss technique. DualityLoss, however, directly compares estimated primal and dual solutions. This difference is the sole source of feedback used to train the pair of neural networks. In contrast, the approach introduced by Shalev-Schwarz and Singer uses labeled examples collected at previous timesteps. Furthermore, Shalev-Schwarz and Singer consider online linear classifiers. DualityLoss is instead a technique for training neural networks in offline settings.

Dual optimization problems have also been used to address intractable problems within the context of GANs. GANs often suffer from a phenomenon called mode collapse. Mode collapse occurs when the generator model stops producing certain classes of outputs. Wasserstein Generative Adversarial Networks (WGANs) are an attempt to address this problem [5]. WGANs use the observation that probability distributions are likely to have non-intersecting support when this support is low-dimensional [4]. This quality is problematic when comparing two distributions using KL divergence because the KL divergence becomes either infinite or zero. WGANs substitute the KL divergence with an Earth-Mover’s distance function. Using Earth-Mover’s distance, however, makes the target optimization intractable. To circumvent this issue, the authors use the technique of Kantorovich-Rubinstein duality to instead optimize the computationally feasible dual problem. WGANs and DualityLoss are similar in that they use duality to address issues surrounding computational complexity. The context of this complexity, however, differs between the
two approaches. In the case of WGANs, the minimax objective function is itself intractable. WGANs use duality to solve an easier problem which shares the same solution. On the other hand, DualityLoss is motivated by the inability to compute data labels. DualityLoss removes the need for such labels by using the estimated duality gap as a measure for solution quality. Unlike WGANs, DualityLoss does not use the dual problem as a substitute for the primal.

2.5 Minimum Cost Flows

The MCF problem and its variants form a class of well-studied problems. The classical formulation consists of a graph equipped with source production values, sink demands, edge capacities, and a linear cost function. The goal is to minimize the flow cost subject to satisfying all demands and abiding by all edge capacities. There are many well-known algorithms, such as a cycle canceling [13] and relaxation [8], which solve this problem. Furthermore, simplex methods from linear programming solve the linear MCF variant [12].

The MCF formulation studied by this work involves uncapacitated edges, as well as a nonlinear, convex cost function. Bertsekas presents both gradient-based approaches and auction algorithms to solve the convex minimum-cost flow problem [7]. The auction algorithm derived by Bertsekas is weakly polynomial and has a lower bound runtime of $\Omega(N^3)$. This technique is among the most efficient algorithms for the convex MCF problem. This cubic runtime, however, prohibits the auction algorithm from scaling to even larger graphs. Végh [41] describes a strongly polynomial algorithm with runtime of $O(|E|^4 \log |E|)$ for cases when the cost function is quadratic. For cost functions of polynomial degree at least three, Hochbaum has shown that no strongly polynomial approximation algorithm exists [20].

The MCF problem is also frequently studied within the context of concave cost functions. This variant is more challenging its convex counterpart, and it is proven to be NP-Hard by Guisewite and Padalos [16]. Montiero et al. [30] use an ant-colony optimization algorithm to approximate uncapacitated minimum cost flows with concave cost functions. Their approach uses a heuristic local search algorithm to improve intermediate solutions at every optimization step. This ant-colony method extends to handle edge-specific fixed costs.

The graph neural network technique presented in this work represents a distinct approach to the MCF problem. Instead of relying on hand-designed heuristics, this neural network learns flows by training on problem instances. This technique has the advantage of offloading a significant amount of computation to the pre-processing phase of training. At runtime, the trained neural network requires only a single pass through the neural network to compute a solution. Hardware components, such as GPUs, further increase the empirical efficiency of the neural network approach.

3 Notation and Problem Statement

As is the case with many algorithmic problems on graphs, the MCF problem can be concisely stated as a constrained optimization problem. This section presents this problem after defining the notation which will appear in the remainder of this thesis.
3.1 Mathematical Notation

Matrices $\mathbf{B} \in \mathbb{R}^{m \times n}$ and vectors $\mathbf{x} \in \mathbb{R}^n$ are denoted by uppercase and lowercase boldface characters respectively. Scalar values are represented by non-boldface characters. This notation extends to specific elements of both matrices and vectors; the element of $\mathbf{B}$ in row $i$ and column $j$ is represented by $B_{ij}$, and the $i^{th}$ element of $\mathbf{x}$ is denoted by $x_i$. Matrices and vectors are both 1-indexed, meaning that $B_{11}$ is the upper-left value of matrix $\mathbf{B}$ and $x_1$ is the first element of vector $\mathbf{x}$. The symbol $[n]$ is defined as $[n] = \{1, \ldots, n\}$, and $\mathbb{R}_{\geq 0}$ is the set $[0, \infty)$. The $i^{th}$ row of the matrix $\mathbf{B}$ is represented by the symbol $\mathbf{B}_i \in \mathbb{R}^n$. Similarly, the $j^{th}$ column is written as $\mathbf{B}_j \in \mathbb{R}^m$. Furthermore, the vectors $\mathbf{1}_n \in \mathbb{R}^n$ and $\mathbf{0}_n \in \mathbb{R}^n$ are defined as the vectors of all ones and zeros respectively. The matrices $\mathbf{1}_{n \times m} \in \mathbb{R}^{n \times m}$ and $\mathbf{0}_{n \times m} \in \mathbb{R}^{n \times m}$ are defined similarly. Finally, the operation $|| : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^{n+m}$ represents vector concatenation.

The set $S = \{s_i \mid i \in [K]\}$ is defined to be a simplex if both $s_i \geq 0$ for all $s_i \in S$ and $\sum_{i=1}^{K} s_i = 1$. The vector $\mathbf{v}$ is a simplex if its set of elements constitutes a simplex. The matrix $\mathbf{B} \in \mathbb{R}^{n \times n}$ is a Markov Chain if every column of $\mathbf{B}$ is a simplex. A Markov chain $\mathbf{B}$ is irreducible if, for all $i, j \in [n]$, there exists some integer $m_{ij} > 0$ such that $B_{ij}^{m_{ij}} > 0$ where $\mathbf{B}^{m_{ij}}$ is the $m_{ij}^{th}$ power of $\mathbf{B}$.

3.2 Graph Notation

A graph $G = (V, E)$ is a collection of vertices and edges. The number of vertices $|V|$ and number of edges $|E|$ are denoted by $n$ and $m$ respectively. An edge $e \in E$ is an ordered pair of vertices $(u, v)$ for $u, v \in V$. For compatibility with indexing into vector and matrix variables, each distinct vertex $u \in V$ is represented by a number in $[n]$. The relative ordering of such numerical labels is unimportant, and no specific vertex ordering will ever be assumed. The notation $u \in V$ and $u \in [n]$ will be used interchangeably.

The graph $G$’s adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is defined to have the following elements for all $i, j \in [n]$.

$$A_{ij} = \begin{cases} 1 & (i, j) \in E \\ 0 & (i, j) \notin E \end{cases}$$

The sets $S^{(u)}_{\text{out}}$ and $S^{(u)}_{\text{in}}$, as shown below, represent the outgoing and incoming neighbors of the node $u \in [n]$.

$$S^{(u)}_{\text{out}} = \{v \in [n] \mid (u, v) \in E\}$$
$$S^{(u)}_{\text{in}} = \{w \in [n] \mid (w, u) \in E\}$$

A simple path $\gamma \subseteq V$ of length $K$ is a finite, ordered set of distinct vertices $\gamma = \{u_0, u_1, \ldots, u_{K-1}\}$ such that $(u_{i-1}, u_i) \in E$ for all $i \in [K-1]$. A simple cycle is equivalent to a simple path except that its start and end points are equal. The distance between any two nodes $u, v \in V$, denoted by $d(u, v)$, is the length of the shortest unweighted path between $u$ and $v$. The diameter of the graph $G$ is defined as $\text{diam}(G) = \max_{u, v \in V[n]} d(u, v)$.

Finally, graphs are often interchangeably referred to as “networks.” Due to the overlap in parlance with neural networks, graphs will never be referred to as “networks” in this report. The word “network” will be used in the context of neural networks.
3.3 The Minimum Cost Flow Problem

This thesis considers the single commodity, uncapacitated MCF problem with a separable cost function [7]. Formally, let the graph $G = (V, E)$ be a directed graph satisfying the assumption below.

**Assumption 1.** The graph $G$ is strongly connected. That is, for every pair of non-equal vertices $u, v \in V$, there is a simple path $\gamma$ which begins at $u$ and ends at $v$.

Let $V_O \subseteq V$ and $V_D \subseteq V$ be the non-intersecting sets of sources and sinks respectively. The vector $d \in \mathbb{R}^n$ holds the demand values for every node. By convention, $d_i < 0$ for $i \in V_O$ and $d_i > 0$ for $i \in V_D$. The demand vector is assumed to meet the following condition.

**Assumption 2.** The demand vector $d$ must sum to zero, i.e. $\sum_{i=1}^n d_i = 0$.

The function $c_{ij} : \mathbb{R} \rightarrow \mathbb{R}$ represents the cost associated with the flow on edge $(i, j) \in E$. For simplicity, the cost function is assumed to be equivalent for all edges, and the global cost function is represented by $c : \mathbb{R} \rightarrow \mathbb{R}$. The cost function is assumed to have the following properties.

**Assumption 3.** The cost function $c : \mathbb{R} \rightarrow \mathbb{R}$ is both differentiable and convex on the domain $\mathbb{R}_{\geq 0}$. Furthermore, $c(0) = 0$ and $c(x) \geq 0$ for all $x \in \mathbb{R}_{\geq 0}$.

Finally, $X \in \mathbb{R}^{n \times n}$ denotes the flow on each edge $(i, j)$ in an adjacency-matrix-style. The flow on non-existent edges is defined to be zero. With these quantities, the MCF problem can be concisely stated as the constrained optimization problem below.

\[
\begin{align*}
\text{min} & \quad \sum_{i=1}^n \sum_{j=1}^n c(X_{ij}) \\
\text{over} & \quad X \in \mathbb{R}^{n \times n} \\
\text{subject to} & \quad \sum_{k=1}^n X_{ki} - \sum_{j=1}^n X_{ij} = d_i \quad \forall i \in V \\
& \quad X_{ij} \geq 0 \quad \forall (i, j) \in E \\
& \quad X_{ij} = 0 \quad \forall (i, j) \notin E
\end{align*}
\]

3.4 Graph Neural Networks

Message-passing graph neural networks are trainable models which operate on graph structured data [18, 23, 36]. Let $h_i^{(0)}$ be the input features for node $i \in [n]$, and let $h_i^{(t)}$ be the state of node $i$ after the $t^{th}$ message-passing step. The state of node $i$ after the $(t + 1)^{st}$ step is computed using the propagation rule below. These operations are executed for all nodes $i \in [n]$.

\[
\begin{align*}
\mathbf{u}_i^{(t+1)} & = \text{AGG}_{t+1}(h_i^{(t)}, \{h_j^{(t)} \mid j \in \mathcal{N}(i)\}) \\
\mathbf{h}_i^{(t+1)} & = \text{UPDATE}_{t+1}(h_i^{(t)}, \mathbf{u}_i^{(t+1)})
\end{align*}
\]
The symbol $\mathcal{N}(i)$ represents any neighborhood of node $i$. The functions $\text{AGG}_{t+1}$ and $\text{UPDATE}_{t+1}$ may contain trainable parameters which are dependent on the current message passing step. The transformations below show a concrete example of this framework. The variables $W^{(t+1)}$ and $b^{(t+1)}$ are trainable parameters.

$$
\begin{align*}
    u_i^{(t+1)} &= \sum_{j \in S_i^{(t)}} h_j^{(t)} \\
    h_i^{(t+1)} &= \tanh \left( W^{(t+1)} (h_i^{(t)} || u_i^{(t+1)}) + b^{(t+1)} \right)
\end{align*}
$$

This basic model can be extended to handle features such as multiple edge types, and this framework encompasses many graph neural network architectures [18, 23, 25, 42].

4 Training Neural Networks Using Lagrangian Duality

The goal of this work is to create a graph neural network model which learns minimum cost graph flows. This problem is difficult because (1) it is not feasible to generate data labels using existing solvers and (2) the neural network must output valid flows. To address these problems, a novel technique to train neural networks for constrained optimization tasks is developed. From a high-level, this technique, called DualityLoss, uses neural networks to represent the primal and dual optimization problem. The networks then have the joint goal of minimizing the estimated duality gap. This section discusses the motivation and details of this approach.

4.1 Motivating DualityLoss

DualityLoss is based on the idea of the duality gaps in constrained optimization. Consider the concrete example of the maximum flow problem. The maximum flow problem aims to find the largest arc flow from a source vertex $s$ to a sink node $t$ which abides by the given edge capacities. The dual to the maximum flow problem is that of finding the minimum $s-t$ graph cut. The maximum flow problem satisfies strong duality. This means that the optimal flow value is equivalent to the weight of the minimum $s-t$ graph cut. This relationship gives rise to a property useful for verifying solutions. Let $d$ be the value of some valid $s-t$ cut and $p$ be the value of a valid $s-t$ flow. If $p = d$, then strong duality implies that both points are optimal; if the values are unequal, then either the primal or the dual is not optimal. Furthermore, if the difference between $p$ and $d$ is large, then one of these points must be far away from the optimum. Intuitively, finding the maximum flow value $p$ is equivalent to finding the valid flow value $p$ and valid cut weight $d$ which minimize $|p - d|$. 

This relationship between primal and dual problems extends to general constrained optimization problems which exhibit strong duality. Figure (3) visually depicts this idea for the primal problem $P$ with the variable $x$ and corresponding dual problem $D$ with the variable $\lambda$. DualityLoss uses neural networks to model the primal and dual problems, and it introduces the goal of minimizing the estimated duality gap. Let $z$ be some vector holding features of the target problem. DualityLoss uses two neural networks: the first is a model for the primal problem and the second represents the dual. The primal and dual neural networks are...
Figure 3: For convex optimization problems with no duality gap, the difference in cost between feasible primal and dual values is a proxy for the distance to the optimum. The function $P$ represents the primal problem, and the function $D$ represents its dual.

denoted by $g_\theta$ and $h_\phi$ respectively. The goal of these networks is to use the problem data $z$ to predict the optimal primal and dual values. As described within the context of maximum flow, this goal is equivalent to finding model parameters $\theta$ and $\phi$ such that $g_\theta(z) = h_\phi(z)$. Thus, minimizing the absolute distance between these two models will result in both the primal and dual models predicting optimal values.

4.2 Defining DualityLoss for Constrained Optimization Problems

The intuition behind the DualityLoss method leads to a loss function which is used to train the primal and dual neural networks. Consider the optimization problem below as an example, and assume that the problem is feasible and bounded for all $a \in \mathbb{R}$.

$$
\begin{align*}
\min_{x \in \mathbb{R}} & \quad P(x) \\
\text{over} & \quad x \in \mathbb{R} \\
\text{subject to} & \quad f_1(x) \leq a
\end{align*}
$$
Let \( x^*_a \) be the optimal value of the problem for the value \( a \in \mathbb{R} \), and let \( z_a \in \mathbb{R}^r \) be some set of features corresponding to the problem instance defined by \( a \). The goal of using a neural network \( g_\theta : \mathbb{R}^r \to \mathbb{R} \) to solve class of constrained optimization problems involves finding the parameters \( \theta \) which allow \( g_\theta \) to map \( z_a \) to \( x^*_a \). Obtaining such a model is highly useful because it can solve the optimization problem for any \( a \) using only a forward pass through the neural network.

The DualityLoss method trains this neural network \( g_\theta \) by introducing a second neural network, \( h_\phi \), which estimates the optimization problem’s dual. The dual function \( D : \mathbb{R}_{\geq 0} \to \mathbb{R} \) for the example problem is written below.

\[
D(\lambda) = \min_{x \in \mathbb{R}} (P(x) + \lambda (f_1(x) - a))
\]

In the context of the example optimization problem, DualityLoss minimizes the following loss function for the training dataset of \( N \) samples, \( \mathcal{X} = \{z_{a_i} \mid a_i \in \mathbb{R}\}_{i=1}^N \).

\[
L(\theta, \phi) = \frac{1}{N} \sum_{i=1}^N (P(g_\theta(z_{a_i})) - D(h_\phi(z_{a_i})))
\]

This loss function is non-negative and has a minimum of zero assuming the following two conditions hold.

1. The primal and dual models always produce feasible solutions.

2. The optimization problem has a duality gap of zero for every problem instance (i.e. the duality gap is zero for all \( a \in \mathbb{R} \)).

The non-negativity of \( L \) comes from the theorem of weak duality. Namely, for any feasible primal point \( x \) and valid dual point \( \lambda \), the inequality \( D(\lambda) \leq P(x) \) holds. The minimum of zero is a result of strong duality.

To see why the primal and dual models must remain feasible, consider figure (4). For feasible primal and dual values, the duality gap is a proxy for how close the values are to the optimum. This interpretation is possible due to the properties of duality. For values outside the feasible region, however, these properties no longer apply, and the difference in “primal” and “dual” cost has no meaning. This idea is shown by \( g_{\theta_2}(z) \) and \( h_{\phi_2}(z) \) in figure (4). Furthermore, the feasible regions of primal and dual problems ensure that the loss function has a unique minimum of zero; if arbitrary points are allowed, the loss function can become unbounded and the models will fail to learn anything meaningful.

The condition of strong duality is necessary so that the loss for every training sample is relative to the same value, namely zero. To clarify this idea, consider if the problem above has a nonzero duality gap. For non-equal constants \( a \) and \( a' \), the duality gap corresponding to the problem with \( a \) is not necessarily equal to the gap corresponding to the value \( a' \). If these problems have different duality gaps, the two instances will have different optimal loss values. This quality would imply that, during training, higher losses do not imply worse model performance. This idea is incompatible with the optimization of neural networks using stochastic gradient descent.
By defining the loss as the difference between the outputs of trainable models, DualityLoss requires no labeled data samples. The technique instead relies heavily upon the properties of strong duality which state that an optimum will occur when the primal and dual models produce feasible values which are equal.

### 4.3 DualityLoss for Minimum Cost Flows

DualityLoss is useful for the MCF problem because existing algorithms for this problem are inefficient; thus, it is not possible to create a sufficiently large dataset of labeled samples to train a neural network. DualityLoss, however, can train a neural network for the MCF problem without requiring labeled data.

To cast the MCF problem within the context of DualityLoss, let $G$ be a strongly-connected, directed graph, $c : \mathbb{R} \to \mathbb{R}$ be the cost function and, $d_k \in \mathbb{R}^n$ be the demands for the $k^{th}$ training sample for $k \in [N]$. Furthermore, let $g_\theta : (G, d) \to \mathbb{R}^{n \times n}$ and $h_\phi : (G, d) \to \mathbb{R}^n$ be the primal and dual models respectively. The vectors $\theta$ and $\phi$ denote the model parameters. The loss function below defines the DualityLoss training objective for these two models. The function $H : \mathbb{R}^n \to \mathbb{R}$ is the dual function, and section 4.6 discusses the dual problem in detail.

$$L(\theta, \phi) = \frac{1}{N} \sum_{k=1}^{n} \left( \left( \sum_{i=1}^{n} \sum_{j=1}^{n} c(g_\theta(G, d_k))_{ij} \right) - H(h_\phi(G, d_k)) \right)$$

(3)

For this loss function to be well-defined, the MCF problem must obey strong duality and the primal and dual models must maintain feasibility. As stated by the proposition below, the MCF problem displays strong duality if the cost function is convex (assumption 3). Maintaining feasibility is the topic of the next section.

**Proposition 1.** For any graph $G = (V, E)$, convex cost function $c$, and valid demand vector $d \in \mathbb{R}^n$, the MCF problem formed from these components has a duality gap of zero.

**Proof.** By proposition 9.4 of Bertsekas’s *Network Optimization*, the convex MCF problem has a duality gap of zero if there exists a feasible solution to the primal problem [7, Chapter 9]. The existence of such a solution is the direct consequence of theorems 1 and 2 stated in section 4.5.2.

### 4.4 Producing Valid Flows with Neural Networks

DualityLoss only works with the MCF problem if the primal MCF neural network always produces valid flows. The requirement is difficult to meet because the flow constraint is complex; there is no straightforward function which can transform the output of a neural network into a valid flow. There do exist functions, however, that constrain the output of neural networks to a probability distribution. Of these functions, softmax is the most well-known. The ability for neural networks to easily output probability distributions leads to a reparameterization of the MCF problem which enables a neural network to always produce valid flows.
Figure 5: Example problem which displays how flow proportions (left) correspond to actual flow values (right). Blue nodes denote sources and red nodes denote sinks.

Instead of directly predicting flow values, the primal model instead outputs relative flow proportions. The proportion on each edge \((i, j)\) represents the fraction of inflow to node \(i\) which should exit on edge \((i, j)\). These proportions sum to one across the outgoing edges of each node, and this feature enforces the conservation of flow at every node. Unlike the flow constraint in the MCF problem, the simplex constraint is easily enforced by softmax, so the primal neural network can easily output valid flow proportions.

Flow proportions can be transformed into actual flow values through a transformation which repeatedly pushes flow according until a fixed point is reached. Formally, let \(P \in \mathbb{R}^{n \times n}\) be a matrix flow proportions produced by the primal model \(g_\theta\). The element \(P_{ij}\) is the amount of inflow to node \(i\) which should exit on edge \((i, j)\). Based on the flow constraint in equation (1c), the function \(f : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^{n \times n}\) below defines a transformation which has a fixed point corresponding to a valid flow. This resulting flow follows the proportions of \(P\).

\[
f(X)_{ij} = \max \left( P_{ij} \left( \sum_{k=1}^{n} X_{ki} - d_i \right), 0 \right) \quad \forall i, j \in [n] \tag{4}
\]

This iterative application of \(f\) is fully described in algorithm 1, and figure 5 displays an example problem. The fixed point iteration of \(f\) provably converges as long as the matrix \(P\) satisfies the conditions of the following remark. This discussion of convergence is left for section 4.5.

**Remark 1.** A flow proportions matrix \(P\) satisfies the two conditions below.

1. \(\sum_{j=1}^{n} P_{ij} = 1 \ \forall i \in [n]\)
2. \(P_{ij} > 0 \ \text{if} \ (i, j) \in E, \ \text{and} \ P_{ij} = 0 \ \text{if} \ (i, j) \notin E\)

The primal model uses both the softmax operator and the graph’s adjacency matrix to satisfy these conditions. Therefore, the primal model can construct a flow proportion matrix which results in a valid flow. This reparameterization successfully allows the primal model to meet the feasibility condition of the primal-dual training approach.
Algorithm 1 Iterative Flow Computation

1: procedure Flow-Solver($P, d, n, \epsilon$) 
2: \hspace{1em} $X^{(0)} \leftarrow 0_{n \times n}$ 
3: \hspace{1em} $X^{(-1)} \leftarrow \text{diag}(\epsilon + 1) \in \mathbb{R}^{n \times n}$ 
4: \hspace{1em} $D \leftarrow 1_n \otimes d \in \mathbb{R}^{n \times n}$ 
5: \hspace{1em} $t \leftarrow 1$
6: \hspace{2.2em} while $\|X^{(t-1)} - X^{(t-2)}\|_1 > \epsilon$ do \hspace{1.2em} $\triangleright$ Matrix 1-norm is entry-wise 
7: \hspace{2.5em} $\bar{X} \leftarrow \left((X^{(t-1)})^T \cdot 1_n\right) \otimes 1_n$
8: \hspace{2.5em} $X^{(t)} \leftarrow P \odot (\bar{X} - D)$ \hspace{1.2em} $\triangleright$ Maximum operator is element-wise 
9: \hspace{2.5em} $X^{(t)} \leftarrow \max(X^{(t)}, 0)$ 
10: \hspace{2.7em} $t \leftarrow t + 1$
11: \hspace{2em} return $X^{(t)}$

4.5 Theoretical Properties of the Iterative Flow Solver

For completeness, this section outlines the derivation of algorithm 1 and states the theorems which ensure the correctness of the iterative process.

4.5.1 Derivation

The goal of a transformation from a proportions matrix $P$ to a flow matrix $X$ is to have $X$ satisfy the property below.

$$X_{ij} = P_{ij} \left(\sum_{k=1}^{n} X_{ki} - d_i\right) \text{ and } X_{ij} \geq 0 \quad \forall i, j \in [n]$$  \hspace{1em} (5)

By ignoring the non-negativity for now, this equation is equivalent to the following.

$$X_{i} = \left(\sum_{k=1}^{n} X_{ki} - d_i\right) \cdot P_{i} \quad \forall i \in [n]$$

The sum $\sum_{k=1}^{m} X_{ki}$ is the sum of the $i$-th column of $X$. Therefore, $\sum_{k=1}^{m} X_{ki} = (X^T 1_n)_i$. The entire flow matrix $X$ can thus be written as follows. The matrix $\bar{X} \in \mathbb{R}^{n \times n}$ is the matrix where each column is $X^T \cdot 1_n$, and $D \in \mathbb{R}^{n \times n}$ is the demand matrix where each row is $d^T$. The symbol $\odot$ denotes the element-wise (Hadamard) product.

$$X = P \odot (\bar{X} - D)$$

The matrix $\bar{X}$ is equivalent to $(X^T 1_n) \otimes 1_n$ where $\otimes$ denotes the vector outer product. Similarly, $D = 1_n \otimes d$. Therefore, a fixed point of the function $f$ (as defined in equation (4)), yields the desired matrix $X$. The element-wise maximum in $f$ ensures that a fixed point meets the non-negativity constraint of equation (5). This derivation shows that a fixed point of $f$ satisfies the original goal in equation (5).
4.5.2 Properties

The reparameterization of the primal problem relies on both the correctness and the convergence of algorithm 1. The theorems guaranteeing these properties are stated in this section. These claims assume the flow proportions matrix $P$ satisfies remark 1. All formal proofs are included in section 10.

Theorem 1, stated below, guarantees that a fixed point of $f$ is indeed a valid flow.

**Theorem 1.** Let $X^* \in \mathbb{R}^{n \times n}_{\geq 0}$ be a fixed point of $f$. Then, $X^*$ satisfies the constraints of the MCF problem in (1).

*Proof.* See section 10.1. □

The convergence of algorithm 1 to a fixed point is guaranteed under theorem 2. Before this theorem is stated, the following notation is defined.

**Definition 1.** The function $f_t$ is function formed by composing $f$ with itself $t$ times. More specifically, for some $X \in \mathbb{R}^{n \times n}_{\geq 0}$, $f_t(X) = f(f(...f(X)))$.

**Theorem 2.** The sequence $(f_t(0_{n \times n}))_{t \in \mathbb{N}}$ converges to a fixed point. This statement implies that the sequence $(X^{(t)})_{t \in \mathbb{N}}$ converges element-wise to a fixed point, where each $X^{(t)}$ is defined by algorithm 1 for $t \geq 0$.

*Proof.* See section 10.3. □

These theorems ensure that the algorithm 1 will always find a valid flow for a given proportions matrix $P$ which satisfies remark 1.

The final consideration of this iterative process involves differentiability. The introduced reparameterization allows the primal neural network to produce the flow proportions $P$ instead of the actual flows $X$. The cost associated with the matrix $P$, however, is still the cost of the corresponding flow matrix. For the primal model to be trainable using the actual flow cost, algorithm 1 must be differentiable with respect to $P$. This statement amounts to showing that the function below is differentiable with respect to $P_{ij}$ for all $i, j \in [n]$.

$$\tilde{f}(P_{ij}) = \max(P_{ij} \sum_{k=1}^{n} X_{ki} - d_i, 0)$$

By defining the derivative of $\max(a, 0)$ at $a = 0$ to be zero, the function $\tilde{f}$ is differentiable with respect to $P_{ij}$. Therefore, loss signals can backpropagate through algorithm 1.

4.6 The Minimum Cost Flow Dual Problem

The goal of the primal-dual training algorithm is to minimize the estimated duality gap. Within the context of MCFs, leveraging this technique requires a neural network which
Algorithm 2 Gradient Descent to solve for Dual Flows

1: procedure DUAL-FLOW-SOLVER(α, n, c, A, β, γ, ϵ) \>
A is the graph’s adjacency matrix

2: Y\(^{(0)}\) ← 0\(_{n×n}\)

3: Y\(^{(-1)}\) ← diag(ϵ + 1) ∈ \(\mathbb{R}^{n×n}\)

4: \(u_{ij}^{(0)}\) ← 0.1 \(\forall i, j \in [n]\)

5: \(t \leftarrow 0\)

6: while \(\|Y^{(t)} - Y^{(t-1)}\|_1 > ϵ\) do \>
Matrix 1-norm is entry-wise

7: for \(i, j \in [n]\) do

8: \(g_{ij}^{(t)} \leftarrow c'(Y_{ij}^{(t)}) + α_j - α_i\) \>
Derivative of the minimization in eq 8

9: \(u_{ij}^{(t+1)} \leftarrow β \cdot u_{ij}^{(t)} + (1 - β) \cdot (g_{ij}^{(t)})^2\)

10: \(Y_{ij}^{(t+1)} \leftarrow \max\left(A_{ij} \cdot \left(Y_{ij}^{(t)} - \frac{γ}{√u_{ij}^{(t+1)} + ϵ}g_{ij}^{(t)}\right), 0\right)\)

11: \(t \leftarrow t + 1\)

12: return \(Y^{(t)}\)

estimates the MCF dual problem. The function below is the Lagrangian of the MCF problem. The vector \(α \in \mathbb{R}^n\) represents the dual variables for each node.

\[
\mathcal{L}(X, α) = \sum_{i=1}^{n} \sum_{j=1}^{n} c(X_{ij}) + \sum_{i \in V} α_i \left(\sum_{k=1}^{n} X_{ki} - \sum_{j=1}^{n} X_{ij} - d_i\right)
\]

\[
= \sum_{i=1}^{n} \sum_{j=1}^{n} (c(X_{ij}) + (α_j - α_i)X_{ij}) + \sum_{i=1}^{n} α_id_i
\]

(6)

(7)

The dual variable \(α_i\) is interpreted as the price of node \(i \in [n]\) [7]. From this Lagrangian, the dual function \(H : \mathbb{R}^n \rightarrow \mathbb{R}\) is defined below. The matrix \(X\) is renamed \(Y\) for clarity.

\[
H(α) = \min_{Y \succeq 0} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} (c(Y_{ij}) + (α_j - α_i)Y_{ij}) + \sum_{i=1}^{n} α_id_i\right)
\]

(8)

The dual neural network can directly produce the dual variables \(α \in \mathbb{R}^n\) because these quantities are unconstrained. Integrating the dual model within the primal-dual loss function (3) requires evaluating the function \(H\). The dual cost function \(H\) is evaluated using projected gradient descent on the matrix \(Y\). As the cost function \(c\) is both convex and differentiable, this process will converge to the required minimum. Algorithm 2 outlines this gradient descent process. The algorithm is an an application of RMSProp [19, 35] to the specific minimization in equation (8). Algorithm 2 is also differentiable with respect to \(α\). This differentiability enables loss signals to backpropagate to the dual model through this algorithm.

Algorithms 1 and 2 define a differentiable processes to evaluate feasible primal and dual MCF solutions respectively. By always producing feasible solutions, the primal and dual formulations presented in this section can be trained using the DualityLoss technique.
5 Graph Neural Network Design

The MCF problem has practical applications in transportation systems. As such, a graph neural network model to compute MCFs should perform well on urban road graphs. Road topologies present a challenge to graph neural networks because their high diameter [27] requires the neural network to retain information over long distances. To address this problem, the graph neural network model used to solve MCFs employs the concepts of graph attention [42], gated state updates [25], and neighborhood lookahead [2]. The graph neural network is integrated into the DualityLoss framework to enable training without labeled data. The entire model is implemented in Tensorflow [1], and the code can be found in the following repository: https://github.com/tejaskannan/graph-primal-dual.

5.1 Motivation and Architecture Overview

The task of learning graph flows can be broken down into two fundamental actions: path-finding and flow-pushing. To be effective at both actions, nodes must know both the location and the demands of the source and sink vertices. To ensure all nodes receive this information, the graph neural network must execute a number of message-passing steps roughly equal to $\text{diam}(G)$ [26]. The experiment displayed in figure 6 empirically confirms this assertion. This chart compares the estimated duality gap for two variants of the same model architecture. The variant which uses a single message passing step is unable to make much progress in minimizing the duality gap. This higher loss translates to flow costs which are 112.5% larger on average. This result shows that nodes require more than just local information for the MCF task.

This observation makes it difficult to apply graph neural networks to high-diameter road...
Figure 7: The 0th (left), 1st (middle), and 2nd (right) order neighborhoods from the vertex A. Red and blue nodes correspond to outgoing and incoming neighborhoods respectively. Vertices with both colors are members of both neighborhoods. Vertex D is not included in the 2nd order outgoing neighborhood because it is already a 1st order outgoing neighbor.

graphs [27]. High-diameter graphs cause issues with long-range information retention. This problem can be addressed by a propagation step which combines information from multiple neighborhood levels [2, 43]. An example of such neighborhoods is shown in figure 7. Information for incoming and outgoing neighbors are separately aggregated. The use of many neighborhoods reduces the total number of message-passes required to cover the entire graph. To further enhance the model’s ability for long-range information retention, nodes use a GRU [11] update to integrate the features collected from the various neighborhoods [25].

Finally, flow values can be interpreted as the emphasis placed on a path from source to sink. This interpretation lends itself to an attention-based model [6, 42] in which nodes can focus on neighbors which should carry more flow. The model presented in this thesis uses multiple attention levels. The first is used to combine node states within a single neighborhood set. The second level is used to aggregated information between the neighborhood levels.

5.2 End-to-End Model

To solve MCF problems, the graph neural network can integrate with the DualityLoss training technique. This integration yields the architecture in figure 8. The input node features are composed of both a trainable and a problem-specific portion. The trainable embeddings allow the model to fit the target graph. The problem-specific information enables the model to adapt to the current flow problem instance. The primal and dual models share the graph neural network component. This decision reduces the number of trainable parameters. Based on the results of Prates et al. [34], the graph neural network component should have the capacity to create useful embedding for both the primal and dual tasks [34]. Separate decoder networks produce specific primal and dual outputs. The loss function is the primal-dual objective established in section 4.3. This function is stated below for completeness. The function $F$ represents the transformation from flow proportions to flow values (algorithm 1), $H$ is the dual cost function, $k$ denotes the training sample index, and
Figure 8: High-level model architecture. Blue node features are trainable embeddings. The red node feature is the demand value for a specific MCF problem instance.

\[ N \] is the size of the training dataset.

\[
L(\theta, \phi) = \frac{1}{N} \sum_{k=1}^{n} \left( \left( \sum_{i=1}^{n} \sum_{j=1}^{n} c(F(g_\theta(G, d_k)))_{ij} \right) - H(h_\phi(G, d_k)) \right)
\]

5.3 Implementation

This section covers the implementation details of the model motivated in the previous sections. In terms of notation, variables named \( W, a, \) and \( b \) are trainable. The vector \( d \) represents the demand vector of a single problem instance.

5.3.1 Encoder

The initial node features are a concatenation of trainable node embeddings with the demand values of a given MCF problem. The initial state of node \( i \in [n] \) is \( y'_i = (y_i || d_i) \) where \( y_i \in \mathbb{R}^{r_1} \) is trainable. The encoder network is a multi-layer perceptron (MLP), and its output is the initial demand-specific embedding \( h_i^{(0)} \in \mathbb{R}^{r_2} \). The transformation below shows the operations for a single dense layer.

\[
h_i^{(0)} = \tanh(W_1 y'_i + b_1)
\]

5.3.2 Graph Attention and Neighborhood Lookahead

The graph neural network component uses the ideas of neighborhood expansion, graph attention, and gated state updates. Formally, let \( A \in \mathbb{R}^{n \times n} \) be the graph’s adjacency matrix and let \( A^k \) be the \( k^{th} \) power of \( A \). The \( k \)-th order outgoing and incoming neighborhoods of
each node \(i \in [n]\) are defined below for all \(k > 1\). For \(k = 0\), \(N^{\text{out}}_k(i) = N^{\text{in}}_k(i) = \{i\}.

\[
N^{\text{out}}_k(i) = \{j \in [n] \mid A^k_{ij} > 0 \text{ and } j \notin N^{\text{out}}_\ell(i) \forall \ell < k\}
\]
\[
N^{\text{in}}_k(i) = \{j \in [n] \mid (A^\top)^k_{ij} > 0 \text{ and } j \notin N^{\text{in}}_\ell(i) \forall \ell < k\}
\]

An example of these neighborhoods is shown in figure 7.

Let \(h_{i}^{(t)}\) be the state of node \(i \in [n]\) after the \(t^{th}\) message-passing step. Furthermore, let \(T\) be the total number of message passes and \(K\) be highest-order neighborhood considered. During the \((t+1)^{st}\) step, nodes use the technique of graph attention [42] to combine information from their various neighborhoods. The equations below define this aggregation for both incoming and outgoing neighborhoods.

\[
\alpha_{ij}^{(k)} = \frac{\exp\{\mathbf{a}_{t}^{\top}(W_{2}h_{i}^{(t)}\|W_{2}h_{j}^{(t)})\}}{\sum_{\ell \in N^{\text{out}}_k(i)} \exp\{\mathbf{a}_{t}^{\top}(W_{2}h_{i}^{(t)}\|W_{2}h_{\ell}^{(t)})\}}
\]
\[
\beta_{ij}^{(k)} = \frac{\exp\{\mathbf{a}_{t}^{\top}(W_{3}h_{i}^{(t)}\|W_{3}h_{j}^{(t)})\}}{\sum_{\ell \in N^{\text{in}}_k(i)} \exp\{\mathbf{a}_{t}^{\top}(W_{3}h_{i}^{(t)}\|W_{3}h_{\ell}^{(t)})\}}
\]
\[
u_i^{(k)} = \tanh \left( \sum_{j \in N^{\text{out}}_k(i)} (\alpha_{ij}^{(k)} W_{2} h_{j}^{(t)}) + b_2 \right)
\]
\[
u_i^{(k)} = \tanh \left( \sum_{j \in N^{\text{in}}_k(i)} (\beta_{ij}^{(k)} W_{3} h_{j}^{(t)}) + b_3 \right)
\]

The vectors \(u_i^{(k)}\) and \(v_i^{(k)}\) represent the aggregated states for the \(k^{th}\) outgoing and incoming neighborhoods. These vectors are in \(\mathbb{R}^{r^2}\). Different trainable weights are applied to the two neighborhood types. This choice follows the common practice of handling different edge types [25]. A second attention level combines the aggregated states for each neighborhood level. This layer is shown below.

\[
\gamma_i^{(k)} = \frac{\exp\{\mathbf{a}_{t}^{\top} u_i^{(k)}\}}{\sum_{j=0}^{K} \exp\{\mathbf{a}_{t}^{\top} u_j^{(j)}\}}
\]
\[
\delta_i^{(k)} = \frac{\exp\{\mathbf{a}_{t}^{\top} v_i^{(k)}\}}{\sum_{j=0}^{K} \exp\{\mathbf{a}_{t}^{\top} v_j^{(j)}\}}
\]
\[
z_{i}^{\text{out}} = \tanh \left( \sum_{j=0}^{K} (\gamma_{i}^{(k)} u_{j}^{(j)}) + b_4 \right)
\]
\[
z_{i}^{\text{in}} = \tanh \left( \sum_{j=0}^{K} (\delta_{i}^{(k)} v_{j}^{(j)}) + b_5 \right)
\]

The vectors \(z_{i}^{\text{out}}, z_{i}^{\text{in}}\) are also in \(\mathbb{R}^{r^2}\). The incoming and outgoing neighborhood states are combined using a single dense layer.

\[
z_{i} = \tanh \left( W_{4} (z_{i}^{\text{out}}\|z_{i}^{\text{in}}) + b_6 \right)
\]

The vector \(z_{i}\) represents an aggregated state based on all neighborhood levels at timestep \(t\). This information is combined with the current node state, \(h_{i}^{(t)}\), using the following GRU update [11, 25]. The symbol \(\sigma\) is the sigmoid function.

\[
q_{i}^{(t+1)} = \sigma \left( W_{q} (h_{i}^{(t)}\|z_{i}) \right)
\]
\[
r_{i}^{(t+1)} = \sigma \left( W_{r} (h_{i}^{(t)}\|z_{i}) \right)
\]
\[
\hat{h}_{i}^{(t+1)} = \tanh \left( W_{s} ((r_{i}^{(t+1)} \odot h_{i}^{(t)}))\|z_{i}) \right)
\]
\[
h_{i}^{(t+1)} = (1 - q_{i}^{(t+1)}) \odot h_{i}^{(t)} + q_{i}^{(t+1)} \odot \hat{h}_{i}^{(t+1)}
\]
Figure 9: Graph Neural Network Architecture. The vector $v^t$ is the state for a single vertex at timestep $t$.

These transformations define how node states are updated between message-passing steps. Figure 9 depicts this entire process. Trainable parameters are shared across all nodes, neighborhood levels, and message-passing steps.

### 5.3.3 Decoder

The graph neural network layer produces a final state $h_i^{(T)}$ for all nodes $i \in [n]$. These states are decoded into primal and dual solutions using distinct MLPs. The transformations for the primal (top) and dual (bottom) networks are shown below for each node $i \in [n]$.

$$s_j^{(i)} = a^T_{primal} \left( \tanh \left( W_{primal} h_i^{(T)} || h_j^{(T)} \right) + b_{primal} \right) \quad \forall j \in [n]$$

$$q_i = a^T_{dual} \left( \tanh \left( W_{dual} h_i^{(T)} + b_{dual} \right) \right)$$

The dual variables $q$ are used to compute the dual cost. The edge scores $s_j^{(i)}$ for all $i, j \in [n]$ are converted to flow proportions as discussed in the next section.

### 5.4 Converting Edge Scores into Flow Proportions

The flow proportions matrix $P$ is created by converting each set $S_i = \{ s_j^{(i)} | (i, j) \in E \} \forall i \in [n]$ into a simplex. The operation can be done by applying the softmax operator to each set $S_i$. The adjacency matrix $A$ masks the scores of non-existent edges. Concretely,
the resulting proportions matrix $P$ has following elements.

$$P_{ij} = \frac{A_{ij} \cdot \exp\{s_j^{(i)}\}}{\sum_{k=1}^{n} A_{ik} \cdot \exp\{s_k^{(i)}\}} \quad i, j \in [n]$$

Softmax has the downside of producing dense distributions [29]. Within the context of MCFs, this property can be undesirable because sending flow along too many paths can lead to suboptimal solutions.

Sparsemax [29] is an alternative to softmax which addresses the issue of dense distributions. Sparsemax works as follows. Let $z \in \mathbb{R}^n$ be the target vector. Then,

$$\text{sparsemax}(z)_i = \max(z_i - \tau(z), 0)$$

where $\tau(z)$ is the unique threshold such that $\sum_{i=1}^{n} \max(z_i - \tau(z), 0) = 1$. This threshold can be computed in a differentiable manner [29]. The key feature of sparsemax is that all values of $z_i$ below this threshold are set to exactly zero. Sparsemax, however, does not guarantee that the proportions matrix $P$ satisfies remark 1. This guarantee is made by assigning all edges at least some near-zero proportion. If $\tilde{z} = \text{sparsemax}(z)$, then

$$\text{clipped-sparsemax}(\tilde{z})_i = \frac{\max(\tilde{z}_i, \epsilon)}{\|\max(\tilde{z}, \epsilon)\|_1}$$

The maximum in the denominator is element-wise and $\epsilon \approx 10^{-9}$ is a small, positive constant. This normalization works because $\tilde{z}$ is a non-negative vector which sums to one. Applying the clipped version of sparsemax to each $S_i$ yields a valid flow proportions matrix $P$, and the qualities of sparsemax imply that it is more aggressive than softmax at assigning edges a near-zero flow proportion.

These two normalizing transformations yield two variants of the proposed graph neural network: Neighborhood+Softmax and Neighborhood+Sparsemax.

### 6 Evaluation

To evaluate the proposed graph neural network solution to the MCF problem, the constructed models are tested on a set of real-world road graphs from Open Street Map [9, 32]. This evaluation has the following five goals. Unless specified, the graph neural network is trained using the DualityLoss technique.
1. **Goal:** Compare the proposed graph neural network approach to the MCF problem against pure optimization algorithms in terms of both answer quality and wall-to-wall running time.

   **Hypothesis:** The neural network will produce flows that are sub-optimal because the pure optimization techniques are allowed many optimization steps on each problem instance. The learned model, however, will produce answers in a more timely manner.

2. **Goal:** Assess the impact of using the DualityLoss method to replace training with ground-truth labels. This comparison encompasses both answer quality and neural network training time, where the training time includes the time to create datasets.

   **Hypothesis:** A neural network trained with DualityLoss will produce slightly higher flow costs than those of the neural network trained with ground truth labels. This hypothesis is based on the fact that the DualityLoss model has no access to ground-truth information. The DualityLoss neural network, however, will be faster to train because it does not require dataset creation.

3. **Goal:** Evaluate the effects of softmax and sparsemax in terms of answer quality.

   **Hypothesis:** Sparsemax will produce lower-costing flows because it can more easily concentrate flow along optimal paths.

4. **Goal:** Compare the Neighborhood architecture against a design which only aggregates information from direct neighbors.

   **Hypothesis:** On high-diameter graphs, the Neighborhood model will produce better answers than this baseline architecture. This improved performance should come from the benefit of using fewer message-passing steps with neighborhood expansion.

5. **Goal:** Assess the importance of the flow proportions matrix $P$ to the flows computed by the iterative flow solver (algorithm 1). This experiment is important to confirm that the results obtained by the neural networks are not solely due to any ingenuity in the iterative flow solver.

   **Hypothesis:** The flow solver will be sensitive to the flow proportions matrix because it blindly follows any given flow proportions. Therefore, the neural network methods will produce lower-costing flows.

### 6.1 An Illustration of Computed Graph Flows

To show that the proposed neural network method is capable of predicting flows, we first consider an example of flows produced by a trained graph neural network model. Figure 11 shows the flows computed by both the neural network and a baseline trust region optimization algorithm [21] for the cost function $c(x) = x^2$. From the figure, it can be seen that the neural network learns the correct overall trend. For example, it transmits flow along the shortest path from source 15 to sink 25. The neural network, however, does face some issues with not fully eradicating flow cycles and overshooting the amount of flow sent to a sink. These issues are explored further in section 7.

The graph neural network used to produce this flow is trained to adapt to changing...
Figure 11: Depiction of flows computed by both the Neural Network and the Trust Region optimizer on the SF-500 graph. These flows are based on the cost function \( c(x) = x^2 \). The thickness of edge \((u, v)\) represents the sum of flows sent on both \((u, v)\) and \((v, u)\). The arrow is placed in the direction of the larger flow. If no reverse edge exists, then the thickness is based only on \((u, v)\). The edge hue denotes the fraction of flow displayed on \((u, v)\) which is actually transmitted on the reverse edge \((v, u)\).
Figure 12: Median flow costs relative to the Trust region optimizer on 1,000 test instances. Error bars denote 25\textsuperscript{th} and 75\textsuperscript{th} percentiles. The ground truth model is not evaluated with the cost of $e^x - 1$ due to the time required to generate data labels. The Neural Network models use the Neighborhood+Sparsemax architecture.

demand patterns for the same set of sources and sinks on the road graph shown in figure 11. More information about the datasets and specifics about model training is included in the section below.

6.1.1 Datasets and Training

The graph neural network method is evaluated on four distinct road graphs obtained from Open Street Map (OSM) [9, 32]. Figure 10 lists some relevant statistics of these graphs. The training, validation, and testing datasets are created by randomly generating demand values for a fixed graph, set of sources, and set of sinks. The neural network is tasked with adapting to different demand patterns. Sources and sinks are chosen such that their pairwise distance is maximized. This setup tests the neural network’s ability to learn longer paths. Appendix 11.2 contains the exact selection algorithm. Demand values are normalized between 0 and 1. To avoid overfitting, test and validation demands are at a distance of at least 0.1 from any training sample. Appendix 11.3 contains all model hyperparameters. The model with the best validation loss is selected for testing.

6.2 Comparing the Neural Network Approach to Baseline Techniques: Flow Cost

The graph neural network approach developed in this work is compared against two pure optimization techniques: sequential least squares quadratic programming (SLSQP) and trust region constrained optimization (Trust Region). The Trust Region algorithm is an advanced optimization technique which produces nearly-optimal values. The Trust Region algorithm is interpreted as the state-of-the-art solver. SLSQP provides lower-quality solutions but can compute such answers more efficiently. Additional background information on these algorithms is included in section 6.6.1.
<table>
<thead>
<tr>
<th>Graph</th>
<th>Cost</th>
<th>Median Runtime (sec) per Problem Instance</th>
<th>Training Time (hrs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pure Optimization</td>
<td>Neural Network</td>
<td>Neural Network</td>
</tr>
<tr>
<td></td>
<td>Trust Region</td>
<td>SLSQP</td>
<td>DualityLoss</td>
</tr>
<tr>
<td>CB-500</td>
<td>$x^2$</td>
<td>0.235s</td>
<td>0.024s</td>
</tr>
<tr>
<td>CB-500</td>
<td>$e^x - 1$</td>
<td>0.609s</td>
<td>0.018s</td>
</tr>
<tr>
<td>SF-500</td>
<td>$x^2$</td>
<td>1.527s</td>
<td>0.091s</td>
</tr>
<tr>
<td>SF-500</td>
<td>$e^x - 1$</td>
<td>1.396s</td>
<td>0.055s</td>
</tr>
<tr>
<td>London-500</td>
<td>$x^2$</td>
<td>N/A</td>
<td>1.428s</td>
</tr>
<tr>
<td>NYC-1000</td>
<td>$x^2$</td>
<td>N/A</td>
<td>7.107s</td>
</tr>
</tbody>
</table>

Figure 13: Execution and training times for both the optimization baselines and the neural network models. The median running times are taken over 1,000 problem instances. Each model is trained for 100 epochs, and the Ground Truth training time includes the time required to generate the training dataset. The neural network uses the Neighborhood+Sparsemax architecture. Omitted entries are excluded due to the high cost of computation.

The second baseline directly evaluates the DualityLoss technique by training a graph neural network using ground-truth labels. This model uses the same Neighborhood architecture but omits the dual decoder component. Labels are composed of the cost values found by the Trust Region optimizer. The ground truth model is not evaluated on the $e^x - 1$ cost function due to the time required to generate a sufficient number of data labels.

Figure 12 depicts a comparison of computed flow costs between the optimization and neural network approaches. As answer quality is related to how close a solution is to the optimum, these costs are displayed relative to the Trust Region optimizer. The chart shows that optimization approaches consistently produce flows which are of lower cost than those of the learned models. The DualityLoss model produces flows within 35% of the optimal value. This result supports the hypothesis regarding the answer quality of the neural network model. The neural network’s suboptimality is explored further in section 7.

When comparing the DualityLoss ground-truth models, the DualityLoss model appears to produce lower-costing flows. Over the 1,000 test instances, the DualityLoss model predicts flows of lower cost 80.3% of the time on CB-500 and 58.6% of the time on SF-500. This result is an encouraging validation of the DualityLoss method. One hypothesis for why DualityLoss achieves better performance is because the DualityLoss model explicitly represents the dual problem. This means that it creates vertex embeddings which contain information about both the primal and dual MCF problems. The primal and dual problems are highly related, and obtaining information about the dual problem can aid in solving with the primal. The relationship between primal and dual MCF problems is exemplified by auction algorithms, as these algorithms solve the MCF problem by directly using the dual [7, Chapter 9]. DualityLoss may therefore have the benefit of creating node embeddings which contain more information about the MCF problem. Exploring this hypothesis in detail is left for future work.
6.3 Comparing the Neural Network Approach to Baseline Techniques: Computation Time

One of the goals of this project is to create a neural network method which can solve the minimum cost flow problem faster than existing techniques. This goal is addressed by measuring the median wall-to-wall time\textsuperscript{1} required to compute flows for both the pure optimization and the neural network methods. Figure 13 shows these measurements. In concordance with the original hypothesis, the neural network methods consistently produce answers faster than both SLSQP and Trust Region. The impact of this efficiency becomes more evidence on the larger London-500 and NYC-1000 graphs. The median runtime of DualityLoss doubles when transitioning from the SF-500 graph to the London-500 graph; on the other hand, SLSQP’s runtime is over 15 times greater.

The price of the neural network’s efficiency at runtime is paid for in its training time. Even for the smaller graphs, the DualityLoss neural network requires over 2 hours of training time. The neural network, however, shows the ability to offload a large amount of the computation to the training phase, and it performs well at testing time.

The final result from this table is the comparison between the DualityLoss and Ground Truth models in terms of training time. When including the time to generate data labels into account, the ground truth model becomes significantly more expensive to train. This high cost is mainly due to the time required to create data labels. The DualityLoss method, on the other hand, requires no labeled datasets, and this benefit is seen in the large discrepancy in training time. Furthermore, the ground truth technique is not computationally feasible on the larger datasets because of the trust region algorithm’s inefficiencies. As proven in this experiment, DualityLoss addresses this problem of scalability and allows for the neural network to be trained on the larger graphs.

![Figure 14: Flow costs for trained Neighborhood models using sparsemax and softmax. These costs are relative to the costs computed by the Trust Region algorithm.](image)

\textsuperscript{1}Times recorded on a Microsoft Azure NC6 Instance with 6 vCPUs, 56GB memory, and a NVIDIA Tesla K80 GPU.
6.4 Evaluating the Design Choices of the Graph Neural Network Architecture

The Neighborhood model makes a few design choices based on both the target problem and the properties of the underlying graph. One such choice is the use of either sparsemax or softmax to convert edge scores into flow proportions. The use of sparsemax is motivated by its ability to more aggressively set small flow proportions. This choice is evaluated on the CB-500 and SF-500 graphs. Figure 14 displays the flow costs of both the Neighborhood+Sparsemax and Neighborhood+Softmax model relative to the Trust Region costs. In three out of the four experiments, the sparsemax model produces slightly lower flow costs. Furthermore, on the SF-500 graph, the sparsemax model predicts better flows on 62.6% test instances for $x^2$ and 63.6% problems for $e^x - 1$. On the CB-500 graph, the sparsemax network predicts lower costing flows on 55.4% and 39.2% of instances for $x^2$ and $e^x - 1$ respectively. Although sparsemax generally performs better, the lack of consistency in these values indicates the sparsemax is not conclusively the better model. This finding goes against the original hypothesis regarding these two operators.

To evaluate design choices behind the Neighborhood model’s architecture, the Neighborhood model is compared to a baseline model called Gated GAT. Gated GAT is built using only the graph attention and gated state update portions of the Neighborhood model’s propagation step. Section 6.6.2 covers the details of this model. The comparison between the Neighborhood model and Gated GAT has the goal of showing that neighborhood expansion allows the model to better compute flows on high-diameter graphs. This goal is assessed by training both the Neighborhood and Gated GAT models on the larger London-500 and NYC-1000 datasets. Figure 15 displays the results of these experiments. On these two datasets, the Neighborhood model outperforms the Gated GAT baseline approach. Over this set of test instances, the Neighborhood model predicts lower-costing flows 99.5% of the time on London-500 and 86.6% of the time on NYC-1000. This discrepancy in figures is an intriguing result because the London-500 graph has a larger diameter than that of NYC-1000. This result indicates that the use of neighborhood expansion allows the neural network to
<table>
<thead>
<tr>
<th>Cost Function</th>
<th>Baseline</th>
<th>Median Increase</th>
<th>Median % Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x^2$</td>
<td>Random</td>
<td>726.9</td>
<td>33,977.5%</td>
</tr>
<tr>
<td></td>
<td>Uniform</td>
<td>232.3</td>
<td>10,466.3%</td>
</tr>
<tr>
<td>$e^x - 1$</td>
<td>Random</td>
<td>4,988.1</td>
<td>50,118.7%</td>
</tr>
<tr>
<td></td>
<td>Uniform</td>
<td>518.7</td>
<td>5,017.5%</td>
</tr>
</tbody>
</table>

Figure 16: Flow cost results for fixed baselines when compared against the Neighborhood+Sparsemax model on the SF-500 graph.

better adapt to higher-diameter graphs.

6.5 Assessing the Impact of the Iterative Flow Solver

The neural network model relies on an iterative flow solver (algorithm 1) to convert learned flow proportions into actual flow values. As this solver plays a critical role in the flow computation process, it is important to ensure that neural network’s results are not simply due to some intelligence present in iterative flow solver. In other words, it must be confirmed that the iterative flow solver does not produce high quality flows for every set of flow proportions.

To assess the abilities of the flow solver, a trained Neighborhood+Sparsemax model is compared to two baselines. The Random baseline feeds a randomly generated valid flow proportions matrix into the flow solver. The Uniform baseline distributes flow across outgoing edges in a uniform manner. Figure 16 shows the flow costs generated by these baselines on the SF-500 dataset\(^2\). These numbers are relative to the Neighborhood+Sparsemax neural network. From this figure, it is evident that flow solver does not always produce low-costing flows. Furthermore, proportions which lead to high-costing flows are easy to find. This analysis confirms the hypothesis that the iterative solver is not capable of turning any set of flow proportions into a low-costing flow. These results give confidence to the notion that the answer quality shown by the neural networks is indeed due to the learned flow proportions.

6.6 Details of the Baseline Solvers

For completeness, this section includes details on the baseline approaches which are used during the evaluation.

6.6.1 Optimization Algorithms

SLSQP [24] is line search technique to iteratively solve constrained optimization problems using a second-order linear approximation. SLSQP considers optimization problems of the

\(^2\)The results on other graphs are similar, and these figures are omitted for brevity.
form below. The functions \( f, g_i \) and \( h_j \) are from \( \mathbb{R}^n \) to \( \mathbb{R} \).

\[
\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t.} \\
g_i(x) = 0 \quad \forall i \in [q] \\
h_j(x) \geq 0 \quad \forall j \in [r]
\]

At each timestep, SLSQP updates its estimate of \( x \) using the 2nd-order Taylor approximation as shown below. The matrix \( H_f(x^{(t)}) \) is the Hessian of \( f \) evaluated at \( x^{(t)} \), and \( \gamma \) is an optional step size parameter.

\[
x^{(t+1)} = x^{(t)} + \gamma \arg \min_{d \in \mathbb{R}^n} \frac{1}{2} d^\top H_f(x^{(t)}) d + \nabla f(x^{(t)}) + f(x^{(t)}) \quad \text{s.t.} \\
g_i(x^{(t)}) + \nabla g_i(x^{(t)})^\top d = 0 \quad \forall i \in [q] \\
h_j(x^{(t)}) + \nabla h_j(x^{(t)})^\top d \geq 0 \quad \forall j \in [r]
\]

Trust region techniques differ from line-search methods in that they optimize over a neighborhood of the current estimate \( x^{(t)} \) [44]. If the region minimizes the function at the next iteration, then the neighborhood is expanded; otherwise, its size is decreased. There are many specific types of trust region algorithms. One such technique is an extension of SLSQP presented by Byrd, Schnabel, and Shultz [10]. This algorithm updates estimates \( x^{(t)} \) as shown below. The parameter \( \theta_t \in [0,1] \) is used to handle the possibility the constraints being violated within the trust region, and \( \Delta_t \) controls the size of the trust region.

\[
x^{(t+1)} = x^{(t)} + \gamma \arg \min_{d \in \mathbb{R}^n} \frac{1}{2} d^\top H_f(x^{(t)}) d + \nabla f(x^{(t)}) + f(x^{(t)}) \quad \text{s.t.} \\
\theta_t g_i(x^{(t)}) + \nabla g_i(x^{(t)})^\top d = 0 \quad \forall i \in [q] \\
\theta_t h_j(x^{(t)}) + \nabla h_j(x^{(t)})^\top d \geq 0 \quad \forall j \in [r] \\
\|d\| \leq \Delta_t
\]

Trust region techniques form a broad class of algorithms, and Yuan [44] provides a good overview of these algorithms and their convergence properties.

The baselines used in this work are from the SciPy optimize library [21]. Each method solves the following optimization problem. This problem is equivalent to the MCF problem in section 3.3.

\[
\min_{x \in \mathbb{R}^n} \sum_{i=1}^{m} c(x_i) \quad \text{such that} \\
B x = d \\
x_i \geq 0 \quad \forall i \in [m]
\]

The matrix \( B \in \mathbb{R}^{n \times m} \) encodes the flow circulation constraint in (1c) and is defined below. In this notation, \( E_k \) refers to the \( k^{th} \) edge for some fixed but arbitrary ordering of \( E \). The
symbol $E_k[0]$ is the source vertex and $E_k[1]$ the destination vertex for the edge $E_k$.

$$b_{ik} = \begin{cases} 
1 & E_k[0] = i \\
-1 & E_k[1] = i \\
0 & \text{else}
\end{cases}$$

Since the target vector $x \in \mathbb{R}^m$ only represents valid edges, the existence constraint (1e) can be ignored. The experimental parameters for both optimization algorithms are shown in figure 11.4.

### 6.6.2 Gated GAT Baseline

The Gated Graph Attention Network (Gated GAT) [25, 42] baseline uses the same principles of the Neighborhood model except for the neighborhood expansion. Explicitly, Gated GAT uses the propagation step below. In terms of notation, $S_{\text{out}}^{(i)}$ and $S_{\text{in}}^{(i)}$ are the sets $S_{\text{out}}^{(i)} \cup \{i\}$ and $S_{\text{in}}^{(i)} \cup \{i\}$ respectively.

$$\alpha_{ij} = \frac{\exp \{a_{\text{out}}^T(W_{\text{out}}h_i^{(t)}||W_{\text{out}}h_j^{(t)})\}}{\sum_{k \in S_{\text{out}}^{(i)}} \exp \{a_{\text{out}}^T(W_{\text{out}}h_i^{(t)}||W_{\text{out}}h_k^{(t)})\}}$$

$$\beta_{ij} = \frac{\exp \{a_{\text{in}}^T(W_{\text{in}}h_i^{(t)}||W_{\text{in}}h_j^{(t)})\}}{\sum_{k \in S_{\text{in}}^{(i)}} \exp \{a_{\text{in}}^T(W_{\text{in}}h_i^{(t)}||W_{\text{in}}h_k^{(t)})\}}$$

$$u_i^{(t+1)} = \tanh \left( \sum_{j \in S_{\text{out}}^{(i)}} (\alpha_{ij}W_{\text{out}}h_k^{(t)} + b_{\text{out}}) \right)$$

$$v_i^{(t+1)} = \tanh \left( \sum_{j \in S_{\text{in}}^{(i)}} (\beta_{ij}W_{\text{in}}h_k^{(t)} + b_{\text{in}}) \right)$$

The vector $u_i^{(t+1)}$ is used to create the updated state $h_i^{(t+1)}$ through the GRU [11, 25] layer below.

$$q_i^{(t+1)} = \sigma \left( W_q(h_i^{(t)}||u_i^{(t+1)}) \right)$$

$$r_i^{(t+1)} = \sigma \left( W_r(h_i^{(t)}||u_i^{(t+1)}) \right)$$

$$\tilde{h}_i^{(t+1)} = \tanh \left( W_s((r_i^{(t+1)} \odot h_i^{(t)}))||u_i^{(t+1)}) \right)$$

$$h_i^{(t+1)} = (1 - q_i^{(t+1)}) \odot h_i^{(t)} + q_i^{(t+1)} \odot \tilde{h}_i^{(t+1)}$$

This baseline differs from the end-to-end Neighborhood model for only the graph neural network component.

### 7 Exploring the Causes of Suboptimality

The empirical evaluation shows that the graph neural network learns flows which are not optimal. With the goal of understanding this suboptimality, this section presents an exploration into the behavior of the graph neural network method. This behavior highlights some issues which lead to non-optimal solutions.
7.1 Adapting to Changes in Demand

The graph neural network has the goal of adapting to different demand patterns. Figure 17 displays an example of this adaptability on the CB-500 graph. In these two examples, the graph neural network shows the ability to coordinate the actions across nodes. This ability is specifically demonstrated by vertices 4 and 8. In the left graph, the largest sink is node 25, and node 4 has a small production value. To cover the demand for node 25, node 8 sends part of its flow along its downward edge. This action is necessary because node 4 cannot cover node 25 on its own. In the right-hand graph, node 4 is the largest source and can satisfy the demand of node 25. It is counterproductive for node 8 to also send to node 25. Node 8 sending flow along its downward edge is suboptimal for two possible reasons.

1. The flow from node 8 may travel to node 25. Along the way, it will combine with the flow from node 4, and the higher edge flow on these arcs will lead to a higher cost. Furthermore, node 4 can already cover node 25. This action by node 8 is thus incurring cost without helping satisfy demand.

2. The flow from node 8 may come back to node 8. To meet all demand, Node 4 must send part of its flow to node 20, and node 8 is on node 4’s shortest path to node 20. As this scenario involves a cycle, the action necessarily leads to a higher cost. If the flow never comes back, then node 4 must have transmitted to node 20 using a non-optimal path.

Node 8 should thus send flow along its upward edge to help cover the remaining two sinks. This strategy is the behavior adopted by the graph neural network. Node 8 executes...
Figure 18: Example flow proportions computed by the neural network on a section of the CB-500 graph. The red path highlights the shortest path from source 15 to sink 20. The blue labels denote the source and sink demands.

contrasting strategies in these examples based on the quantities at nodes 4 and 25. These strategies highlight the graph neural network’s ability to both coordinate node actions and adapt to demand patterns. The ability is encouraging because the neural network learns this behavior without ever observing ground-truth labels.

7.2 Causes of Suboptimal Flows

The graph neural network produces non-optimal flows for a few key reasons. The first cause of suboptimality is the overallocation of sinks. Consider the graph in figure 18. This graph is the upper region of the left-hand graph in figure 17. The labels on each edge are the flow proportions computed by the graph neural network. From these proportions, node 15 sends 39.6% of its production value along its shortest path to the sink 20 (highlighted in red). This set of proportions results in 0.312 units of flow entering 20. This strategy is an over-allocation to the sink. Recovering from this error requires this excess flow to make its way to an unsaturated sink. This recovery generally incurs a high cost because this excess flow is now itself originating from a sink. Since the proportions are static, they are not updated based on a sink’s saturation level. Therefore, vertices such as node 21 will continue to feed flow into sink 20. The problem of overallocation also leads to higher penalties is higher on larger graphs because the paths between sinks are longer. In this example, the ideal case calls for node 15 to send at most 32.9% of its source value to node 20. This figure is in contrast with the 39.6% instructed by the graph neural network. Techniques for overcoming this 7% difference are crucial to producing truly optimal flows.

The second main cause of suboptimality involves graph cycles. For an increasing cost
Figure 19: Example of flow proportions assigned to source-sink paths which do not lead to a valid flow. The quantities on each edge represent flow proportions.

function, sending flow around a cycle leads to a higher total cost. Based on vertices 21 and 17 in figure 18, the neural network fails to completely eradicate flow cycles. During training, the neural network is never explicitly prohibited from sending flow around cycles. Instead, the model must learn to eradicate cycles because they cause higher costs. Cycles, however, play a key role in maintaining feasibility. For example, in the case the sink over-allocation in figure 18, some of the excess flow into node 20 will eventually leak out to node 17. Node 17 will then transmit some of this flow to unsaturated sinks. If cycles are completely eradicated, mistakes such as over-allocation lead to non-feasible outputs. This behavior occurs because a weakly connected flow graph can cause flow to get trapped in areas with no unsaturated sinks. In this manner, cycles are key to maintaining feasibility yet lead to suboptimal answers. This tension likely makes it difficult for the graph neural network to eliminate all cycles.

7.3 Mitigating Flow Cycles with Pre-computed Paths

The role occupied by cycles leads to a larger question about the proposed representation of the MCF problem. The graph neural network model proposed in this thesis learns graph flows without any explicit prevention of cycles. If the model were designed to avoid cycles, however, then the learned flows would likely be of lower cost. One possible strategy to prevent cycles is to only transmit flow along precomputed simple paths. Instead of having to learn paths directly, the model can instead focus on determining the optimal quantity of flow to send between each source and sink. Furthermore, since each precomputed path is simple, flows constrained to these paths are less likely to result in unnecessary cycles. This strategy has the further benefit of eliminating the need for an iterative flow solver.

This idea, however, induces problems regarding feasibility because the neural network can still over-allocate flow to a single sink. Figure 7.3 is such a situation. In this example, both sinks receive 0.5 units of flow, and this allocation leaves C unsaturated. This issue of feasibility can be numerically solved by projecting the neural network’s output onto the subspace of feasible solutions. This idea, however, creates a break between the neural network’s output and the final solution. Nevertheless, simplifying the learning task to explicitly
remove cycles should help encourage the neural network to never pass flow around a cycle.

8 Future Work

The task of learning graph flows presents some interesting avenues for future work. Two such extensions are discussed below.

8.1 Capacity Constraints

Adding support for capacity constraints makes the neural network model more applicable to real-world scenarios. One way to add this feature is to introduce a barrier term to the cost function. By incurring a high cost when an edge capacity is violated, the barrier incentivizes the neural network to abide by the capacities. A second method involves introducing slack variables and augmenting the target graph as proposed in section 4.1.2 of Bertsekas’s Network Optimization [7, Chapter 4]. This approach allows the neural network model to maintain feasibility using the technique of learning flow proportions. This method, however, comes at the cost of doubling the number of nodes and edges in the target graph.

8.2 Transfer Learning

The model presented in this work uses random initial node embeddings. Graph flow problems, however, are highly related to simpler problems such as shortest paths. This fact is exemplified by the correlation between flow and betweenness centrality measures [31]. One approach to potentially accelerate the primal-dual learning process is to use initial embeddings which are pre-trained on tasks such as shortest paths. As algorithms for the shortest-path problem efficient, this pre-training phase can leverage ground-truth labels. The correlation between these tasks may enable the knowledge of shortest paths to transfer to learning graph flows.

9 Conclusion

This thesis introduces new a technique to learn uncapacitated MCFs using graph neural networks. This technique leverages a novel training method, called DualityLoss, which removes the need for labeled data and instead relies on principles of Lagrangian duality. This quality is important within the context of MCFs because existing algorithms cannot efficiently produce large labeled datasets required for training. DualityLoss uses neural networks to estimate the primal and dual optimization problems, and these networks are jointly trained with the objective of minimizing the estimated duality gap. This technique encourages the two models to reach their respective optima when the target problem exhibits strong duality and the models produce feasible answers. The end-to-end graph neural network is demonstrated by learning minimum cost flows on a set of real-world road graphs. These results show that, despite never using ground-truth labels during training, the model both learns flows which are close to being optimal and computes graphs flows far faster than existing techniques.
10 Proofs

This section contains the formal proofs of theorems 1 and 2. These proofs assume the target graph $G$ is strongly connected (assumption 1). The function $f$ is defined in equation (4). The matrix $P \in \mathbb{R}^{n \times n}$ refers to a matrix of flow proportions satisfying remark 1. The vector $d \in \mathbb{R}^n$ is the vector of demands under assumption 2.

10.1 Correctness

Theorem 1. Let $X^* \in \mathbb{R}^{n \times n}_{\geq 0}$ be a fixed point of $f$. Then, $X^*$ satisfies the constraints of the MCF problem in equation (1).

Proof. From the definition of a fixed point, the following relationship holds for all $i, j \in [n]$.

$$X^*_{ij} = \max(P_{ij}(\sum_{k=1}^{n} X^*_{ki} - d_i), 0)$$

This matrix is clearly non-negative. It also satisfies $X^*_{ij} = 0$ if $(i, j) \notin E$ because $P_{ij} = 0$ if $(i, j) \notin E$. It remains to show that the flow constraint in equation (1c) holds for this matrix $X^*$.

For now, let us omit the maximum operator and assume that $X^*_{ij} = P_{ij}(\sum_{k=1}^{n} X^*_{ki} - d_i)$. We want to show that $\sum_{k=1}^{n} X^*_{ki} - \sum_{j=1}^{n} X^*_{ij} - d_i = 0$ for all $i \in [n]$. This relationship is shown below. Step 3 relies on the fact that $\sum_{j=1}^{n} P_{ij} = 1$ (true by remark 1).

$$\sum_{k=1}^{n} X^*_{ki} - \sum_{j=1}^{n} X^*_{ij} - d_i = \sum_{k=1}^{n} X^*_{ki} - \left(\sum_{j=1}^{n} P_{ij}(\sum_{q=1}^{n} X^*_{qi} - d_i)\right) - d_i$$

$$= \sum_{k=1}^{n} X^*_{ki} - \sum_{j=1}^{n} P_{ij} \left(\sum_{q=1}^{n} X^*_{qi}\right) + \sum_{j=1}^{n} P_{ij} d_i - d_i$$

$$= \sum_{k=1}^{n} X^*_{ki} - \sum_{q=1}^{n} X^*_{qi} + d_i - d_i$$

$$= 0$$

To justify the exclusion of the maximum operator, we must show that $P_{ij}(\sum_{k=1}^{n} X^*_{ki} - d_i) \geq 0$ for all $i, j \in [n]$. For the sake of contradiction, assume there exists some $s, t \in [n]$ such that $P_{st}(\sum_{k=1}^{n} X^*_{kt} - d_t) < 0$. Using the definition of a fixed point, we have the following.

$$\sum_{i=1}^{n} \sum_{j=1}^{n} X^*_{ij} = \sum_{i=1}^{n} \sum_{j=1}^{n} f(X^*)_{ij}$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{n} \max\left(P_{ij}(\sum_{k=1}^{n} X^*_{ki} - d_i), 0\right)$$

$$\geq \sum_{i=1}^{n} \sum_{j=1}^{n} P_{ij}(\sum_{k=1}^{n} X^*_{ki} - d_i)$$
From the construction of $s$ and $t$, we have that $P_{st}(\sum_{k=1}^{n} X_{kt} - d_t) < 0$. This fact implies that the maximum operator is strictly increasing the value of at least one $(i,j)$ term (namely $(s, t)$). Therefore, the inequality above must be strict. This observation yields the following.

$$\sum_{i=1}^{n} \sum_{j=1}^{n} X_{ij} > \sum_{i=1}^{n} \sum_{j=1}^{n} P_{ij} \left( \sum_{k=1}^{n} X_{ki} - d_i \right)$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{n} P_{ij} \cdot \left( \sum_{k=1}^{n} X_{ki}^* \right) - \sum_{i=1}^{n} \sum_{j=1}^{n} P_{ij} d_i$$

$$= \sum_{i=1}^{n} \left( \sum_{k=1}^{n} X_{ki}^* \right) \sum_{j=1}^{n} P_{ij} - \sum_{i=1}^{n} d_i \sum_{j=1}^{n} P_{ij}$$

$$= \sum_{i=1}^{n} \sum_{k=1}^{n} X_{ki}^* - \sum_{i=1}^{n} d_i$$

$$= \sum_{i=1}^{n} \sum_{k=1}^{n} X_{ki}^*$$

The fourth equality holds because $\sum_{j=1}^{n} P_{ij} = 1$ (remark 10, and the final equality is true since $\sum_{j=1}^{n} d_i = 0$ (assumption 2). The analysis above shows that $\sum_{i=1}^{n} \sum_{j=1}^{n} X_{ij}^* > \sum_{i=1}^{n} \sum_{k=1}^{n} X_{ki}^*$. This statement is a contradiction because both sides are a sum over the same terms. Therefore, no such $s, t \in [n]$ exist, so $P_{ij}(\sum_{k=1}^{n} X_{ki}^* - d_i) \geq 0$ for all $i, j \in [n]$. This completes the proof that $X^*$ satisfies all constraints of the MCF problem. 

### 10.2 Continuity

Before we prove the convergence of the sequence $(f_t(X))_{t \in \mathbb{N}}$, we will show that $f$ is continuous. This property is important because, if the sequence $(f_t(X))_{t \in \mathbb{N}}$ converges, the continuity of $f$ implies the existence of a fixed point of $f$. The continuity of $f$ is rigorously shown below.

**Proposition 2.** The function $f : \mathbb{R}^{n \times n}_{\geq 0} \rightarrow \mathbb{R}^{n \times n}_{\geq 0}$ is element-wise continuous.

**Proof.** Let $\epsilon > 0$ be arbitrary and let $X, Y \in \mathbb{R}^{n \times n}_{\geq 0}$ be matrices such that $\|X - Y\|_1 < \delta = \epsilon$. The norm $\|\cdot\|_1$ is the element-wise 1-norm. We want to show that $\|f(X) - f(Y)\|_1 < \epsilon$. This fact is shown in the steps below. This analysis uses an alternative formula for the maximum, $\max(a, b) = \frac{1}{2}(a + b + |a - b|)$. The absolute value from $P_{ij}$ is omitted because these values are non-negative (remark 1).

$$\|f(X) - f(Y)\|_1 = \sum_{i=1}^{n} \sum_{j=1}^{n} |\max(P_{ij}(\sum_{k=1}^{n} X_{ki} - d_i), 0) - \max(P_{ij}(\sum_{k=1}^{n} Y_{ki} - d_i), 0)|$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{1}{2} P_{ij} \left| \sum_{k=1}^{n} X_{ki} - d_i + \sum_{k=1}^{n} X_{ki} - d_i \right| - \sum_{k=1}^{n} Y_{ki} + d_i - \sum_{k=1}^{n} Y_{ki} - d_i$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{1}{2} P_{ij} \left| \sum_{k=1}^{n} (X_{ki} - Y_{ki}) + \sum_{k=1}^{n} X_{ki} - d_i \right| - \sum_{k=1}^{n} Y_{ki} - d_i$$

$$\leq \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{1}{2} P_{ij} \left| \sum_{k=1}^{n} |X_{ki} - Y_{ki}| + \sum_{k=1}^{n} X_{ki} - d_i \right| - \sum_{k=1}^{n} Y_{ki} - d_i$$

$$\leq \epsilon$$

This completes the proof that $X^*$ satisfies all constraints of the MCF problem. 

42
\[
\leq \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{1}{2} P_{ij} \left| \sum_{k=1}^{n} (X_{ki} - Y_{ki}) + \sum_{k=1}^{n} X_{ki} - d_i - \sum_{k=1}^{n} Y_{ki} + d_i \right| \quad \text{(rev. triangle ineq.)}
\]
\[
= \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{1}{2} P_{ij} \left| \sum_{k=1}^{n} (X_{ki} - Y_{ki}) + \sum_{k=1}^{n} (X_{ki} - Y_{ki}) \right|
\]
\[
\leq \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{1}{2} P_{ij} \left| \sum_{k=1}^{n} (X_{ki} - Y_{ki}) + \sum_{k=1}^{n} |X_{ki} - Y_{ki}| \right| \quad \text{(triangle ineq.)}
\]
\[
\leq \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{1}{2} P_{ij} \sum_{k=1}^{n} 2|X_{ki} - Y_{ki}| \quad \text{(triangle ineq.)}
\]
\[
= \sum_{i=1}^{n} \sum_{k=1}^{n} |X_{ki} - Y_{ki}| \cdot \sum_{j=1}^{n} P_{ij}
\]
\[
\leq \sum_{i=1}^{n} \sum_{k=1}^{n} |X_{ki} - Y_{ki}|
\]
\[
= \|X - Y\|_1 < \delta = \epsilon
\]

Therefore, \(\|f(X) - f(Y)\|_1 < \epsilon\) for \(\|X - Y\|_1 < \delta = \epsilon\). This proves that \(f(X)\) is jointly continuous over the entire matrix \(X\). It is then a theorem that \(f\) is also separately (element-wise) continuous. \(\Box\)

### 10.3 Convergence

This section has the goal of showing that the fixed point iteration of \(f\) converges when given a starting point of \(X^{(0)} = 0_{n \times n}\). We first introduce a lemma which is used in the formal proof of convergence. The matrix \(X^{(t)}\) refers to \(f_t(X^{(0)})\).

**Lemma 1.** For all \(t \geq 0\) and \(i, j \in [n]\), \(X_{ij}^{(t+1)} \geq X_{ij}^{(t)}\).

**Proof.** Let \(P(t)\) be the proposition that \(X_{ij}^{(t+1)} \geq X_{ij}^{(t)}\) for all \(i, j \in [n]\). The lemma is equivalent to the statement: \(P(t)\) holds for all \(t \geq 0\). We proceed by induction on \(t\).

**Base Case:** For \(t = 0\), we have the following for all \(i, j \in [n]\).

\[X_{ij}^{(1)} = \max \left(0, P_{ij} \sum_{k=1}^{n} X_{ki}^{(0)} - P_{ij} d_i \right) \geq 0\]

As \(X^{(0)} = 0_{n \times n}\), \(X_{ij}^{(0)} = 0\). We have thus shown that \(X_{ij}^{(1)} \geq X_{ij}^{(0)}\), so \(P(0)\) holds.

**Inductive Step:** Assume that \(P(t)\) is true. We want to show \(P(t + 1)\). Using the definition
\[ X^{(t+1)}_{ij} = \max \left( 0, P_{ij} \sum_{k=1}^{n} X^{(t)}_{ki} - P_{ij} d_i \right) \]
\[ \geq \max \left( 0, P_{ij} \sum_{k=1}^{n} X^{(t-1)}_{ki} - P_{ij} d_i \right) \]
\[ = X^{(t)}_{ij} \]

The second step comes from applying the induction hypothesis. This relationship proves that \( P(t + 1) \) holds.

To prove the convergence of the sequence \((f(X^{(0)}))_{t \in \mathbb{N}}\), we establish a link between algorithm 1 and Markov chains. We note that algorithm 1 is the implementation of the sequence \((f(X^{(0)}))_{t \in \mathbb{N}}\).

Consider a variant of algorithm 1 presented in algorithm 3. Algorithm 3 is based on a Markov-chain-like transformation. This property makes it simpler to analyze. Algorithms 1 and 3 are logically equivalent; rather than consider the flow on each edge, algorithm 3 uses the total inflow to each node for only the current timestep. To formally prove the relationship between the two algorithms, we will show that
\[ \sum_{k=1}^{n} X^{(t)}_{ki} - \sum_{k=1}^{n} X^{(t-1)}_{ki} = y^{(t)}_{i} \]
for all nodes \( i \in [n] \).

**Algorithm 3**

Markov-chain-based Flow Computation

1: procedure MARKOV-CHAIN-FLOW(P, d, n, \( \epsilon \))
2: \( y^{(0)} \leftarrow [-d_i \text{ for } i \in [n] \text{ if } d_i < 0 \text{ else } 0] \)
3: \( r^{(0)} \leftarrow 0_n \)
4: \( r^{(1)} \leftarrow [d_i \text{ for } i \in [n] \text{ if } d_i > 0 \text{ else } 0] \quad \triangleright \text{Remaining sink demands} \)
5: \( t \leftarrow 1 \)
6: while \( \|y^{(t-1)}\|_1 > \epsilon \) do
7: \( \tilde{y}^{(t-1)} \leftarrow \max(y^{(t-1)} - r^{(t-1)}, 0) \quad \triangleright \text{Maximum is element-wise} \)
8: \( y^{(t)} \leftarrow P^\top \cdot \tilde{y}^{(t-1)} \)
9: \( r^{(t+1)} \leftarrow \max(r^{(t)} - y^{(t)}, 0) \)
10: \( t \leftarrow t + 1 \)
11: \( \text{return } \sum_{s=0}^{t} y^{(s)} \)

**Proposition 3.** For all iterations \( t > 0 \),
\[ \sum_{k=1}^{n} X^{(t)}_{ki} - \sum_{k=1}^{n} X^{(t-1)}_{ki} = y^{(t)}_{i} \]
for all nodes \( i \in [n] \) where \( X^{(t)} \) is the matrix in algorithm 1 and \( y^{(t)} \) is the vector in algorithm 3.

**Proof.** We proceed by strong induction on the number of completed iterations, \( t > 0 \).

**Base Case:** Let \( t = 1 \), and let \( i \in [n] \) be arbitrary. By the definition of \( X^{(0)} \), we have
\[\sum_{k=1}^{n} X_{k\ell}^{(0)} = 0 \text{ for all } \ell \in [n].\] This statement implies the following.

\[\sum_{k=1}^{n} X_{ki}^{(1)} = \sum_{k=1}^{n} \max(P_{ki} \sum_{j=1}^{n} X_{jk}^{(0)} - P_{ki}d_k, 0) = \sum_{k=1}^{n} \max(-P_{ki}d_k, 0)\]

Similarly, from algorithm 3, we have the following.

\[y_i^{(1)} = \sum_{k=1}^{n} P_{ki} \tilde{y}_k^{(0)}\]

From the initialization of \(y^{(0)}\) and \(r^{(0)}\), \(\tilde{y}_k^{(0)} = \max(-d_k, 0)\). Thus, \(y_i^{(1)} = \sum_{k=1}^{n} \max(-P_{ki}d_k, 0) = \sum_{k=1}^{n} X_{ki}^{(1)}\) for all \(i \in [n]\).

Inductive Step: Assume the proposition holds for all iterations less than or equal to \(t\). We want to show that it also holds for iteration \(t+1\). Let node \(i \in [n]\) be arbitrary. Using the definition of algorithm 1, we have the following. Step 2 uses the fact that \(\max(a - b, 0) = a - \min(a, b)\).

\[\sum_{k=1}^{n} X_{ki}^{(t+1)} - X_{ki}^{(t)} = \sum_{k=1}^{n} \max(P_{ki} \sum_{j=1}^{n} X_{jk}^{(t)} - P_{ki}d_k, 0) - \max(P_{ki} \sum_{j=1}^{n} X_{jk}^{(t-1)} - P_{ki}d_k, 0)\]

\[= \sum_{k=1}^{n} P_{ki} \left( \sum_{j=1}^{n} X_{jk}^{(t)} - \min(\sum_{j=1}^{n} X_{jk}^{(t)} - P_{ki}d_k) - \sum_{j=1}^{n} X_{jk}^{(t-1)} + \min(\sum_{j=1}^{n} X_{jk}^{(t-1)}, d_k) \right)\]

\[= \sum_{k=1}^{n} P_{ki} \left( \sum_{j=1}^{n} (X_{jk}^{(t)} - X_{jk}^{(t-1)}) - \min(\sum_{j=1}^{n} X_{jk}^{(t)} - P_{ki}d_k) + \min(\sum_{j=1}^{n} X_{jk}^{(t-1)}, d_k) \right)\]

\[= \sum_{k=1}^{n} P_{ki} \left( y_k^{(t)} - (\min(\sum_{j=1}^{n} X_{jk}^{(t)} - P_{ki}d_k) - \min(\sum_{j=1}^{n} X_{jk}^{(t-1)} - P_{ki}d_k)) \right)\] (9)

The last step comes from applying the induction hypothesis. We claim the following.

\[\tilde{y}_k^{(t)} = y_k^{(t)} - (\min(\sum_{j=1}^{n} X_{jk}^{(t)} - P_{ki}d_k) - \min(\sum_{j=1}^{n} X_{jk}^{(t-1)} - P_{ki}d_k))\] (10)

where \(\tilde{y}_k^{(t)} = \max(y_k^{(t)} - r_k^{(t)}, 0)\) is defined in algorithm 3. To gain insight into why this relationship holds, we further specify the residual value \(r_k^{(t)}\) below. The first equality comes
from the definition of \( r^{(t)} \) and the second equality comes from the strong induction hypothesis.

\[
r^{(t)}_k = \max \left( d_k - \sum_{s=0}^{t-1} y^{(s)}_k, 0 \right)
\]

\[
= \max \left( d_k - \left( \sum_{j=0}^{n} X^{(t-1)}_{jk} - \sum_{j=0}^{n} X^{(t-2)}_{jk} + \sum_{j=0}^{n} X^{(t-2)}_{jk} - \ldots + \sum_{j=0}^{n} X^{(1)}_{jk} - \sum_{j=0}^{n} X^{(0)}_{jk} \right), 0 \right)
\]

\[
= \max \left( (d_k - \sum_{j=0}^{n} (X^{(t-1)}_{jk} + X^{(0)}_{jk})), 0 \right)
\]

\[
= \max \left( d_k - \sum_{j=0}^{n} X^{(t-1)}_{jk}, 0 \right)
\]

The last step holds because \( X^{(0)} = 0_{n \times n}. \) We turn our attention back to proving equation 10. There are four possible cases.

1. The first possibility is that the left minimum is greater than the right minimum. This case is impossible as it implies that the flow along an edge decreases with time and thus violates lemma 1.

2. The second scenario is that both minimums result in \( d_k. \) Thus,

\[
\min(\sum_{j=1}^{n} X^{(t)}_{jk}, d_k) - \min(\sum_{j=1}^{n} X^{(t-1)}_{jk}, d_k) = 0
\]

This fact implies that \( \sum_{j=1}^{n} X^{(t-1)}_{jk} > d_k, \) so \( r^{(t)}_k = 0. \) Thus, \( \tilde{y}^{(t)}_k = y^{(t)}_k \) by the definition of \( \tilde{y}^{(t)}_k. \) Using the assumption for this scenario, this equality proves equation (10) in this case.

3. The third case is that the left minimum results in \( d_k \) while the right is \( \sum_{j=1}^{k} X^{(t-1)}_{jk}. \) Equation (10) results in \( y^{(t)}_k - d_k + \sum_{j=1}^{k} X^{(t-1)}_{jk}. \) This term is exactly \( r^{(t)}_k \) because \( d_k \geq \sum_{j=1}^{k} X^{(t-1)}_{jk} \) by assumption. Furthermore, \( y^{(t)}_k \geq r^{(t)}_k \) by the following relationship.

\[
y^{(t)}_k - r^{(t)}_k = \sum_{j=1}^{k} X^{(t)}_{jk} - \sum_{j=1}^{k} X^{(t-1)}_{jk} - d_k + \sum_{j=1}^{k} X^{(t-1)}_{jk}
\]

\[
= \sum_{j=1}^{k} X^{(t)}_{jk} - d_k \geq 0 \quad \text{(by assumption of this case)}
\]

Since \( \tilde{y}^{(t)}_k = \max(y^{(t)}_k - r^{(t)}_k, 0) = y^{(t)}_k - r^{(t)}_k, \) we have shown that \( \tilde{y}^{(t)}_k = y^{(t)}_k - d_k + \sum_{j=1}^{k} X^{(t-1)}_{jk} \) because \( r^{(t)}_k = d_k - \sum_{j=1}^{k} X^{(t-1)}_{jk} \) in this scenario. This proves equation (10) for this case.
4. The final scenario occurs when neither minimum equals $d_k$. The right-hand-side of equation (10) results in the following.

$$y_k^{(t)} - \left( \sum_{j=1}^{n} X_{jk}^{(t)} - \sum_{j=1}^{n} X_{jk}^{(t-1)} \right) = 0$$

This term equals zero due to the induction hypothesis. To prove the claim, it remains to show that $y_k^{(t)} - r_k^{(t)} \leq 0$. This property is displayed below.

$$y_k^{(t)} - r_k^{(t)} = \sum_{j=1}^{k} X_{jk}^{(t)} - \sum_{j=1}^{k} X_{jk}^{(t-1)} - d_k + \sum_{j=1}^{k} X_{jk}^{(t-1)}$$

$$= \sum_{j=1}^{k} X_{jk}^{(t)} - d_k \leq 0 \quad \text{(by the assumption of this case)}$$

Thus, equation (10) holds in this final case.

Through these cases, we have proven that equation (10) holds. Therefore, using equation 9, we obtain the following.

$$\sum_{k=1}^{n} (X_{ki}^{(t+1)} - X_{ki}^{(t)}) = \sum_{k=1}^{n} P_{ki} \tilde{y}^{(t)}$$

$$= \mathbf{P}^\top \cdot \tilde{y}^{(t)}$$

$$= y^{(t+1)}$$

The last equality comes from the definition of $y^{(t+1)}$ on line 8 of algorithm 3. This completes the inductive step and proves that $y^{(t)} = \sum_{k=1}^{n} X_{ki}^{(t)} - X_{ki}^{(t-1)}$ for all $t > 0$. 

Before we formally prove the termination of algorithm 1, we prove two final lemmas.

**Lemma 2.** Let $\mathbf{B} \in \mathbb{R}_{\geq 0}^{n\times n}$ be a Markov chain such that $\sum_{i=1}^{n} B_{ij} = 1$ for all $j \in [n]$. Then, for any $\mathbf{x} \in \mathbb{R}_{\geq 0}^{n}$, $\|\mathbf{Bx}\|_1 = \|\mathbf{x}\|_1$.

**Proof.** From the definition of matrix multiplication, we have the following.

$$\|\mathbf{Bx}\|_1 = \sum_{i=1}^{n} \sum_{j=1}^{n} B_{ij} |x_j|$$

$$= \sum_{j=1}^{n} \sum_{i=1}^{n} B_{ij} |x_j|$$

$$= \sum_{j=1}^{n} |x_j| \sum_{i=1}^{n} B_{ij}$$

$$= \sum_{j=1}^{n} |x_j| = \|\mathbf{x}\|_1$$
Lemma 3. For all iterations \( t \geq 1 \) of algorithm 3, \( \|y^{(t)}\|_1 = \|r^{(t)}\|_1 \).

Proof. We proceed by induction on \( t > 0 \).

Base Case: Let \( t = 1 \). As \( P^T \) is an irreducible Markov chain (remark 1) and \( r^{(0)} = 0 \), we know that \( \|y^{(1)}\|_1 = \|P^T y^{(0)}\|_1 = \|y^{(0)}\|_1 \) by lemma 2. By construction \( \|y^{(0)}\|_1 = D \) and \( \|r^{(1)}\|_1 = D \), so \( \|y^{(1)}\|_1 = \|r^{(1)}\|_1 \). This completes the proof of the base case.

Inductive Step: Let the lemma hold for iteration \( t \). Using the updates for both \( y^{(t+1)} \) and \( r^{(t+1)} \), we get the following. We use the non-negativity of all terms to omit absolute values for the 1-norm.

\[
\|y^{(t+1)}\|_1 = \|Py^{(t+1)}\|_1 \\
= \|y^{(t+1)}\|_1 \quad \text{(Lemma 2)} \\
= \sum_{i=1}^{n} \max(y^{(t)}_i - r^{(t)}_i, 0) \\
= \sum_{i=1}^{n} \frac{1}{2}(y^{(t)}_i - r^{(t)}_i + |y^{(t)}_i - r^{(t)}_i|) \quad \text{(alternative formula for max(a, b))} \\
= \sum_{i=1}^{n} y^{(t)}_i - \frac{1}{2}(y^{(t)}_i + r^{(t)}_i - |y^{(t)}_i - r^{(t)}_i|) \\
= \sum_{i=1}^{n} y^{(t)}_i - \sum_{i=1}^{n} \frac{1}{2}(y^{(t)}_i + r^{(t)}_i - |y^{(t)}_i - r^{(t)}_i|) \quad \text{(induction hypothesis)} \\
= \sum_{i=1}^{n} r^{(t)}_i - \frac{1}{2}(y^{(t)}_i + r^{(t)}_i - |y^{(t)}_i - r^{(t)}_i|) \\
= \sum_{i=1}^{n} \frac{1}{2}(-y^{(t)}_i + r^{(t)}_i + |y^{(t)}_i - r^{(t)}_i|) \\
= \sum_{i=1}^{n} \max(r^{(t)}_i - y^{(t)}_i, 0) \\
= \|r^{(t+1)}\|_1
\]

The last step holds because all terms are non-negative. This shows that the norm of \( y \) is equal to the norm of the residual \( r \) for all iterations.

With these results in hand, we prove that both algorithms terminate for any \( \epsilon > 0 \).

Proposition 4. For any \( \epsilon > 0 \), algorithm 3 will terminate. This result implies that algorithm 1 will also terminate for any \( \epsilon > 0 \).

Proof. To prove that algorithm 3 terminates, it suffices to prove that \( \|y^{(t)}\|_1 \to 0 \) as \( t \to \infty \).

Let \( t_0 \) be some iteration such that there exists an \( i \in [n] \) where \( y^{(t_0)}_i > 0 \). If no such \( t_0 \) exists, then the algorithm must have converged because \( y^{(t)} \) is non-negative. Let \( S \) be the set of
all nodes \( j \in [n] \) such that \( r_{j}^{(t_0)} > 0 \). By lemma 3 and the non-negativity of \( r^{(t_0)} \), we know that \( S \) is nonempty. Let \( t_1 \geq 0 \) be the smallest value such that \( (\langle P^t \rangle(t_1)\tilde{y}^{(t_0)})_j > 0 \) for some \( j \in S \). Such a value \( t_1 \) must exist because \( P^t \) is an irreducible Markov chain. We can appeal to this irreducibility because, by construction, the system behaves like a Markov chain for the iterations between \( t_0 \) and \( t_1 \). This appeal holds by the following argument. Since \( y_k^{(t)} = 0 \) for all \( k \in S \) and \( t \in [t_0, t_0 + t_1] \), lines 7 and 9 of algorithm 3 make no changes to either \( y^{(t)} \) or \( r^{(t)} \) for all \( t \in [t_0, t_0 + t_1] \). This leaves only the transformation \( P^t \) within this timeframe.

After iteration \( t_0 + t_1 + 1 \), we know that \( \|y^{(t_0+t_1+1)}\|_1 < \|y^{(t_0)}\|_1 \) because both \( y_j^{(t_0+t_1)} \) and \( r_j^{(t_0+t_1)} \) are strictly positive by construction. At this point, we can repeat the above process to further decrease the 1-norm of both the inflow vector \( y^{(t)} \) and \( r^{(t)} \). Now, let \((s_k)_{k \in \mathbb{N}}\) be the sequence of steps such that both \( y_j^{(s_k)} \) and \( r_j^{(s_k)} \) are greater than zero for some \( j \in [n] \). We have shown that the sequence \( \|y_j^{(s_k)}\|_1 \) is strictly decreasing. Furthermore, it is bounded below by zero due to the non-negativity of the 1-norm. Therefore, \( (\|y_j^{(s_k)}\|_1)_{s \in \mathbb{N}} \) is a strictly decreasing sequence which is bounded below, so \( \|y_j^{(s_k)}\|_1 \to 0 \) as \( k \to \infty \). Furthermore, we know that \( (\|y^{(t)}\|_1)_{t \in \mathbb{N}} \) is non-increasing since line 7 of algorithm 3 can only decrease the norm of \( y^{(t)} \). The norm is preserved by the transformation on line 8 due to lemma 2. Thus, the limit of the convergent subsequence \( (\|y_j^{(s_k)}\|_1)_{s \in \mathbb{N}} \) implies that \( \|y^{(t)}\|_1 \to 0 \) as \( t \to \infty \) as well. This statement proves that algorithm 3 terminates for any \( \epsilon > 0 \).

We now use the termination of algorithm 3 to conclude that algorithm 1 terminates as well. We can use proposition 3 to obtain the following. We can omit the absolute values due to lemma 1.

\[
\|X_j^{(t)} - X_j^{(t-1)}\|_1 = \sum_{i=1}^{n} \sum_{j=1}^{n} X_{ij}^{(t)} - X_{ij}^{(t-1)} \\
= \sum_{j=1}^{n} \sum_{i=1}^{n} X_{ij}^{(t)} - X_{ij}^{(t-1)} \\
= \sum_{j=1}^{n} y_j^{(t)} = \|y^{(t)}\|_1
\]

Using the fact that \( \|y_j^{(t)}\|_1 \to 0 \) as \( t \to \infty \), we can conclude that \( \|X_j^{(t)} - X_j^{(t-1)}\|_1 \to 0 \) as \( t \to \infty \). This statement shows that the loop in algorithm 1 will terminate for any \( \epsilon > 0 \). \( \square \)

Proposition 4 proves that algorithm 1 will terminate. It does not, however, guarantee that a limit exists for the sequence of flow matrices. Intuitively, we have only shown that the distance between consecutive elements in the fixed point iteration goes to zero; this fact does not guarantee that a limit exists. The convergence of this sequence is stated and proven in the theorem below.

**Theorem 2.** The sequence \( (f_t(0_{n \times n}))_{t \in \mathbb{N}} \) converges to a fixed point. This statement implies that the sequence \( (X_j^{(t)})_{t \in \mathbb{N}} \) converges element-wise to a fixed point, where each \( X_j^{(t)} \) is defined by algorithm 1 for \( t \geq 0 \).

**Proof.** We will prove the theorem by first showing that the series \( \sum_{t=0}^{\infty} y_j^{(t)} \) converges element-wise. The convergence of \( \sum_{t=0}^{\infty} y_j^{(t)} \) will be displayed by a comparison to a geometric series.
Let $t_0 \in \mathbb{N}$ be the first iteration for which the norm of $y^{(t)}$ decreases. This decrease happens when $y^{(t_0)}$ indicates inflow into some sink node which has unmet demand. For all $t < t_0$, each element of $y^{(t)}$ is upper bounded by the total flow production, $D$, since $\|y^{(t)}\|_1 = D$ and $y^{(t)}$ is non-negative. Furthermore, $t_0$ itself is bounded above by $m$, the number of edges in the graph. This bound holds because $t_0$ is equal to the (unweighted) shortest path length between any source and sink with unmet demand. Clearly, $m = |E|$ is the upper bound for a simple path in the graph $G$. We claim that, every $m$ iterations, the norm of $y^{(t)}$ decreases by at least a factor of $(1 - r)$ where $r = \prod_{(i,j) \in E} P_{ij}$. In other words, $\|y^{(km)}\|_1 \leq D(1 - r)^k$ for $k \in \mathbb{Z}_{\geq 0}$. As a note $(1 - r) < 0$ because each $P_{ij} > 0$ for $(i, j) \in E$ by remark 1. We prove this claim by induction on $k \geq 0$.

Base Case: Let $k = 0$. By definition, $\|y^{(0)}\|_1 = D$. Thus, the claim holds for $k = 0$.

Inductive Step: Assume the claim holds for $k$. We want to show that it also holds for $k + 1$. Let $S = \{i \in [n] \mid d_i > 0\}$ be the set of all sinks. Consider the behavior of the algorithm between iterations $mk$ and $m(k + 1)$. Let node $j \in [n]$ be such that $j$ has more incoming flow than outgoing flow at iteration $m$. Formally, $y_j^{(km)} > 0$. Then, $j$ has a path $\gamma$ of length at most $m$ to some sink $i \in S$ with unmet demand. Such a path is guaranteed by the irreducibility of $P$. The proportion of flow which is transferred to $i \in S$ along $\gamma$ is $y_j^{(km)} \cdot \prod_{(u,v) \in \gamma} P_{uv}$. The proportion is nonzero by remark 1. This process occurs for all $j \in [n]$ such that $y_j^{(km)} > 0$. As $y^{(km)}$ is non-negative, the set of all such $j$ is corresponding to a total flow of at most $D(1 - r)^k$ by the induction hypothesis. Thus, in the worst case, all of the flow is concentrated on a single node which is $m$ hops away from the nearest sink. This upper-bound “path” retains exactly $\prod_{(i,j) \in E} P_{ij} = r$ fraction of the given flow. Thus, by iteration $m(k + 1)$, the total unmet demand must have decreased by at least $rD(1 - r)^k$. Therefore, $\|y^{(m(k+1))}\|_1 \leq D(1 - r)^k - rD(1 - r)^k = D(1 - r)^{k+1}$. This completes the inductive proof.

We have shown that $\|y^{(km)}\|_1 \leq D(1 - r)^k$ for all $k \geq 0$. As the vector $y^{(t)}$ is non-negative, we have that each element of $y^{(km)}$ is at most $D(1 - r)^k$ as well. The sequence $(y^{(t)}_i)_{t \in \mathbb{N}}$ for $i \in [n]$ is then bounded above by the sequence $(a_k)_{k \in \mathbb{N}}$ shown below.

$$D, D, \ldots, D, D(1 - r), D(1 - r), \ldots, D(1 - r), D(1 - r)^2, D(1 - r)^2, \ldots$$

Each term is repeated $m$ times. We claim that the series $\sum_{k=0}^{\infty} a_k$ converges. We can redefine the series as $\lim_{N \to \infty} \sum_{k=0}^{N} a_k = \lim_{N \to \infty} \sum_{k=0}^{N} mD(1 - r)^k$. Combining terms in this manner is possible because the number of repeated terms is finite. The limit $\lim_{N \to \infty} \sum_{k=0}^{N} mD(1 - r)^k$ exists because it is a geometric series with ratio $(1 - r) < 1$. Therefore, $\lim_{t \to 0} y^{(t)}_i$ converges for all $i \in [n]$ by comparison with $\sum_{k=0}^{\infty} a_k$. Note that since each term is non-negative, this series converges absolutely.

Using the proof of proposition 3, we know that $\sum_{t=0}^{N} y^{(t)}_i = \sum_{k=1}^{n} X^{(N)}_{ki}$. Therefore, the existence of $\lim_{N \to \infty} \sum_{t=0}^{N} y^{(t)}_i$ means that $\lim_{N \to \infty} \sum_{k=1}^{n} X^{(N)}_{ki}$ must exist as well. We will call this limit $L_i$. Since each $X^{(t)}_i$ is non-negative, the summation over $k$ is finite, and the sequence of these matrices is element-wise non-decreasing by lemma 1, we have that each $X^{(t)}_i \leq L_i$ for all $t \geq 0$ and $i, k \in [n]$. Thus, each $(X^{(t)}_{ki})_{t \in \mathbb{N}}$ is a monotone sequence which is bounded above.
This fact implies that each \((X^{(t)}_k)_{t \in \mathbb{N}}\) converges, so the sequence of these matrices converges element-wise. We have thus shown that the sequence \((f_t(0_{n \times n}))_{t \in \mathbb{N}}\) converges element-wise to some matrix \(X^*\). Since \(f\) is continuous by proposition 2, the matrix \(X^*\) must be a fixed point of \(f\). This statement means that the sequence of matrices in algorithm 1 converges element-wise to a fixed point.

In summary, these proofs have shown that the sequence formed by repeatedly applying \(f\) is guaranteed to converge to a non-negative fixed point which is a valid flow. Therefore, algorithm 1 converges to a valid flow.
References


11 Appendix

11.1 Graphs

The table below shows the Open Street Map (OSM) [9, 32] queries used to obtain the graphs used during evaluation.

<table>
<thead>
<tr>
<th>Graph Name</th>
<th>Target Address</th>
<th>Distance (Meters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CB-500</td>
<td>Pembroke College, Cambridge, UK</td>
<td>500</td>
</tr>
<tr>
<td>SF-500</td>
<td>555 Market Street, San Francisco, USA</td>
<td>500</td>
</tr>
<tr>
<td>London-500</td>
<td>Leicester Square, London, UK</td>
<td>500</td>
</tr>
<tr>
<td>NYC-1000</td>
<td>Empire State Building, New York City, USA</td>
<td>1000</td>
</tr>
</tbody>
</table>

11.2 Algorithm to Generate Sources and Sinks

Algorithm 4 Algorithm to generate sources and sinks.

1: procedure GENERATE-SOURCES-SINKS($G = (V, E)$, nSources, nSinks)
2: $v_0 \leftarrow$ RANDOM-VERTEX($G$)  
3: nodes $\leftarrow \{v_0\}$  
4: while LENGTH(nodes) < (nSources + nSinks) do
5: $u^* \leftarrow \arg\min_{u \in V - \text{nodes}} (\min_{v \in \text{nodes}} (\min (\text{Dist}(G, u, v), \text{Dist}(G, v, u))))$
6: nodes.add($u^*$)
7: sources $\leftarrow$ nodes[0:nSources]
8: sinks $\leftarrow$ nodes[nSources:nSinks]
9: return sources, sinks

11.3 Model Hyperparameters

The table below show the hyperparameters which are held constant over all datasets. All models are trained with the Adam [22] optimizer and use an online batch selection [28] technique.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step Size</td>
<td>0.001</td>
</tr>
<tr>
<td>Gradient Clip</td>
<td>1</td>
</tr>
<tr>
<td>Batch Size</td>
<td>100</td>
</tr>
<tr>
<td>Initial Embedding Dimensions</td>
<td>16</td>
</tr>
<tr>
<td>Hidden State Dimensions</td>
<td>32</td>
</tr>
<tr>
<td>Encoder Units per Layer</td>
<td>(8, 32, 32)</td>
</tr>
<tr>
<td>Decoder Units per Layer</td>
<td>(32, 32, 32, 1)</td>
</tr>
<tr>
<td>Max Dual Iters</td>
<td>500</td>
</tr>
<tr>
<td>Dual Step Size</td>
<td>0.01</td>
</tr>
<tr>
<td>Dual Momentum</td>
<td>0.9</td>
</tr>
</tbody>
</table>
The hyperparameters below refer to those used for the Neighborhood models on each dataset. The Gated-GAT baseline uses the same hyperparameters apart from both the maximum neighborhood level and the number of message-passing steps. For a given graph, the Gated-GAT model is configured to use the following. The variable $\text{mps}_{\text{gated-gat}}$ refers to the number of message-passing steps.

$$\text{mps}_{\text{gated-gat}} = \text{mps}_{\text{nbrhood}} \times \text{max-nbrhood-level}_{\text{nbrhood}}$$

$$\text{attn-heads}_{\text{gated-gat}} = \text{max-nbrhood-level}_{\text{nbrhood}} + 1$$

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Cambridge-500</th>
<th>SF-500</th>
<th>London-500</th>
<th>NYC-1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Flow Iters</td>
<td>1,000</td>
<td>1,000</td>
<td>1,000</td>
<td>1,000</td>
</tr>
<tr>
<td>Message-Passing Steps</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>Max Neighborhood Level</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Training Samples</td>
<td>10,000</td>
<td>10,000</td>
<td>20,000</td>
<td>25,000</td>
</tr>
<tr>
<td>Validation Samples</td>
<td>1,000</td>
<td>1,000</td>
<td>2,000</td>
<td>2,500</td>
</tr>
<tr>
<td>Test Samples</td>
<td>1,000</td>
<td>1,000</td>
<td>2,000</td>
<td>2,500</td>
</tr>
<tr>
<td>No. Sources</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>No. Sinks</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

### 11.4 Configurations for Optimization Algorithms

The configurations for SLSQP (left) and Trust Region Optimization (right) are listed below.

```json
{  
  "max_iter": 1000,
  "tolerance": 1e-7,
  "hessian": "BFGS",
  "jacobian": "2-point",
  "factorization": "SVD"
}
```