# Warm Start Algorithms for Bipartite Matching and Optimal Transport

Akash Mittal

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science & Application

Bo Ji, Chair
Sharath Raghvendra
Thang Hoang

December 18th, 2024
Blacksburg, Virginia

Keywords: Optimal Transport, Bipartite Matching, Primal-Dual Method, Learning Augmented Algorithms, Additive Approximation Algorithms

# Warm Start Algorithms for Bipartite Matching and Optimal Transport

Akash Mittal

(ABSTRACT)

Minimum Cost Bipartite Matching and Optimal Transport are essential optimization challenges with applications in logistics, artificial intelligence, and multimodal data alignment. These problems involve finding efficient pairings while minimizing costs. Due to the combinatorial nature of minimum-cost bipartite matching and optimal transport problems, the worst-case time complexities of standard algorithms are often prohibitively high. To mitigate this, prior information or warm starts are commonly employed to accelerate computation. In this thesis, we propose two novel warm-start algorithms for solving the minimum-cost bipartite matching problem, with the first algorithm extending naturally to the optimal transport problem. The first algorithm uses the $LMR$ algorithm to produce dual weights with respect to a scaled version of an approximate matching or transport plan with high additive error. Using these dual weights as warm starts enables a faster computation of approximate matchings or transport plans with extremely small additive error $\delta$ faster than the original $LMR$ algorithm, which currently holds the state-of-the-art execution time for approximating the optimal transport plan, a generalization of bipartite matching, in the sequential setting. By achieving lower additive error at a faster rate, our method provides a smoother trade-off between exact and approximate matchings or transport plans. Given that $C$ is largest cost of the given matching or transportation problem and $\delta$ is our chosen additive error, the running time for our first warm start algorithm for matching is $\mathcal{O}\left(n^2 \cdot \min\left(\frac{C}{\delta}, \sqrt{n}\right)\right)$ and for optimal transport is $\mathcal{O}\left(n^2 \cdot \min\left(\sqrt{\frac{nC}{\delta}}, \frac{C}{\delta}\right)\right)$. Our second warm-start algorithm leverages

machine-learned weights to accelerate the computation of minimum-cost bipartite matching. This learning-augmented approach achieves a theoretically superior running time compared to the previous best learning-augmented algorithms for the same problem. Overall, this thesis highlights the effectiveness of combining theoretical algorithmic advancements with modern learning-based techniques, resulting in robust and efficient solutions for fundamental optimization problems.

# Warm Start Algorithms for Bipartite Matching and Optimal Transport

Akash Mittal

(GENERAL AUDIENCE ABSTRACT)

Matching problems and optimal transport are foundational tools with wide-ranging applications in fields like logistics, artificial intelligence, and multimodal data analysis. At their core, these problems involve finding the most efficient way to pair objects or resources while minimizing associated costs. Imagine we are tasked with pairing items from two groups based on compatibility. A perfect matching ensures every item is paired exactly once, maximizing compatibility. Often, the challenge extends to minimizing the cost of these matchings, resulting in the **minimum-cost maximum-cardinality matching problem**. This problem is fundamental to both theoretical algorithmic research and real-world applications like resource allocation, task scheduling, and network optimization. **Optimal transport** builds on this idea, where we align resources (e.g., supply nodes) with demands (e.g., demand nodes) in a way that minimizes transportation costs. For instance, consider distributing supplies across a network with varying transport costs. The goal is to fully meet demands while minimizing the overall cost—a crucial requirement in supply chain logistics, training generative models like GANs, and aligning multimodal data such as text and images. This research introduces **warm start algorithms** to accelerate these complex problems by leveraging two strategies:

1. **Machine Learning for Dual Weights:** Predictive models are used to estimate dual weights that guide optimization, reducing the computational complexity of solving the problem.

2. **Simplified Problem Refinements:** Initial solutions are derived from solving simplified sub-problems, which are then iteratively refined for improved precision and efficiency.

By combining algorithmic rigor with predictive and incremental approaches, these methods scale efficiently to large datasets and diverse applications. This work not only advances the theoretical understanding of optimization problems but also can have practical impacts in areas like generative modeling and cross-modal alignment.

# Dedication

*I dedicate this work to my family, for their unwavering support and encouragement during the toughest of times.*

# Acknowledgments

I would like to express my deepest gratitude to my advisors, Dr. Sharath Raghvendra and Dr. Bo Ji, whose invaluable guidance, encouragement, and mentorship have been pivotal throughout my research journey. Their insightful feedback and thoughtful discussions have deepened my understanding of the subject and broadened my academic horizons. Their expertise, patience, and unwavering support have helped me grow as a researcher and made this thesis possible. I am also sincerely thankful to Dr. Thang Hoang for his time, thoughtful feedback, and expert review of my work. Additionally, I would like to acknowledge Pouyan Shirzadian for his insightful discussions and continuous support during the early stages of my research. Finally, I am profoundly grateful to my family and friends, especially my parents, Ashish Mittal and Monica Mittal, and my sister, Aastha Mittal, for their constant emotional and mental support. Their encouragement and belief in me have been a source of strength throughout this journey. From the bottom of my heart, thank you.

# Contents

**List of Figures** **xi**

**1 Introduction** **1**

1.1 Important Concepts . . . . . 2

1.2 Related Work . . . . . 4

1.2.1 Classical Matching Algorithms . . . . . 4

1.2.2 Optimal Transport and Approximation Algorithms . . . . . 5

1.2.3 Learning-Augmented Algorithms . . . . . 6

1.3 Main Results . . . . . 7

**2 Preliminaries** **9**

2.1 Hopcroft-Karp Algorithm . . . . . 10

2.2 Hungarian Algorithm . . . . . 11

2.2.1 Dual Feasibility . . . . . 11

2.2.2 Algorithm . . . . . 13

2.2.3 Invariants . . . . . 14

2.3 Scaling Algorithms . . . . . 14

2.3.1 Gabow's Algorithm . . . . . 15

2.3.2 Gabow and Tarjan's Algorithm . . . . . 17

**3 Scaling Algorithm with Warm Starts** **21**

3.1 *LMR* Algorithm . . . . . 21

3.1.1 Introduction . . . . . 21

3.1.2 Scaling Supplies and Demands . . . . . 25

3.1.3 Algorithm for Scaled Supplies, Demands and Costs . . . . . 26

3.1.4 Constructing Transport Plan for Original Capacities . . . . . 28

3.1.5 Proof of Correctness . . . . . 29

3.1.6 Running Time Analysis . . . . . 30

3.2 Warm Start Algorithm . . . . . 33

3.2.1 Definitions and Notations . . . . . 34

3.2.2 The Algorithm . . . . . 34

3.2.3 Scaling Algorithm . . . . . 35

3.2.4 Adjustment Step: . . . . . 37

3.2.5 Bipartite Matching via Warm Starts . . . . . 37

3.2.6 Correctness Proofs . . . . . 38

3.2.7 Efficiency Proofs . . . . . 40

3.3 Experimental Results . . . . . 50

3.3.1 Dataset and Experiments: . . . . . 50

3.3.2 Results . . . . . 50

**4 Learning Augmented Algorithms** **53**

4.1 Prior Algorithm for Minimum Cost Matching . . . . . 53

4.1.1 Learning Good Dual Weights . . . . . 54

4.1.2 Optimization Algorithm . . . . . 55

4.2 Our Optimization Algorithm . . . . . 56

4.2.1 Running Time Proofs . . . . . 58

**5 Conclusions** **62**

**Bibliography** **64**

# List of Figures

3.1 Delta vs Iterations for Matching: $LMR$ algorithm with warm starts and $LMR$ algorithm. . . . . 51

3.2 Delta vs Iterations for OT: $LMR$ algorithm with warm starts and $LMR$ algorithm. . . . . 51

# Chapter 1

# Introduction

Warm start algorithms are an essential class of techniques in optimization that leverage precomputed or approximate solutions to initialize iterative algorithms closer to the optimal solution. This approach significantly reduces computation time and enhances the efficiency of solving complex problems. Some of the works that use warm starts to accelerate the running time of standard optimization algorithms are as follows [2, 3, 5, 11, 16, 22, 23, 25, 27]. The two problems that we will focus on in this thesis is the Optimal Transport (OT) problem and the Minimum Cost Perfect Bipartite Matching. These problems have significant importance in areas of applications such as computer vision, data analysis, economics and etc [4, 6, 7, 9, 12, 28].

In the context of Optimal Transport, we design and analyze a warm-start method that leverages approximate transport plans represented by precomputed dual weights. This approach facilitates faster convergence toward transport plans with low $\delta$ additive error. In this algorithm intermediate solutions act as feasible initializations for progressively finer levels of precision. By incorporating warm starts, the algorithm avoids the computational expense of starting from scratch at each iteration, leading to significant reductions in runtime.

Similarly, for the minimum cost Bipartite Matching, we will examine and improve warm start algorithms that exploit both machine learned dual weights and computed dual weights from previous intermediate instances to accelerate the matching process and significantly reduces the number of adjustments needed to reach desired matching and dual weights.

The importance of warm start algorithms lies in their ability to balance computational efficiency and accuracy. By integrating approximate solutions or leveraging machine-learned predictions, these algorithms not only achieve faster convergence but also exhibit robust performance across a range of problem instances. As a result, warm start methodologies are starting to become a cornerstone in the design of scalable and practical algorithms for combinatorial optimization problems.

## 1.1 Important Concepts

Graphs are mathematical structures that model relationships between objects, represented as vertices, with connections between them denoted as edges. Formally, a graph $G(V, E)$ consists of a vertex set $V$ and an edge set $E \subseteq V \times V$. In practical applications, the vertices often correspond to real-world entities, while the edges signify relationships or interactions between these entities. This work focuses on leveraging graphs to model and solve matching and transportation problems, particularly those arising in bipartite graphs.

The goal of this work is to design better asymptotic running time algorithms for these fundamental optimization problems. Asymptotic analysis abstracts away implementation-specific constants and focuses on the algorithm's behavior for large inputs, addressing fundamental challenges in problem complexity. This perspective allows us to explore the intrinsic difficulty of matching problems and design techniques that address their core computational barriers.

In this thesis we will work with bipartite graphs. Equivalently, bipartite graphs are characterized by the absence of odd-length cycles. This structural property simplifies matching problems significantly compared to general graphs, making bipartite graphs a natural focus for algorithmic research.

A matching $M \subseteq E$ in a graph is a subset of edges such that no two edges share a common vertex. Matchings represent disjoint pairs. Among the many problems in this domain is the maximum-cardinality matching problem, which seeks the largest possible matching in a graph. Another central problem, the minimum-cost maximum-cardinality matching problem, extends this concept by associating a cost $c(u, v)$ with each edge $(u, v) \in E$ and aims to find a maximum-cardinality matching $M$ that minimizes the total cost, $c(M) = \sum_{(u,v)\in M} c(u, v)$.

Optimal transport is closely related to bipartite matching, serving as a fundamental framework for measuring similarity between datasets or probability distributions. In applications such as training Generative Adversarial Networks (GANs) and multimodal alignment tasks, transportation cost functions are crucial. In GANs, optimal transport cost aligns generated and target distributions, improving sample quality. For multimodal tasks, it aligns data across different modalities (e.g., text and images) by minimizing transport costs between feature spaces.

Formally, the transportation problem involves a complete bipartite graph $G(A, B)$, where $A$ represents demand nodes and $B$ represents supply nodes. Each vertex $a \in A$ has a positive or zero demand $d_a$, and each $b \in B$ has a positive or zero supply $s_b$. The cost of transporting one unit of supply is $c(a, b) \geq 0$, and $C$ denotes the maximum edge cost. Without loss of generality, the total supply does not exceed the total demand: $U = \sum_{b\in B} s_b \leq D = \sum_{a\in A} d_a$.

A transport plan is a function that gives non-negative values $\sigma(a, b)$ to each edge subject to the constraints:

$$\sum_{b\in B} \sigma(a, b) \leq d_a, \quad \text{for all } a \in A, \quad \text{and} \quad \sum_{a\in A} \sigma(a, b) \leq s_b, \quad \text{for all } b \in B.$$

A transport plan is maximum if all supply is fully transported, i.e., $\sum_{a\in A} \sigma(a, b) = s_b$ for all

$b \in B$. The cost of a transport plan $\sigma$, denoted $w(\sigma)$, is given by:

$$w(\sigma) = \sum_{(a,b)\in A\times B} \sigma(a,b) \cdot c(a,b).$$

The aim is to derive the minimum-cost maximum transport plan. Given the minimum-cost maximum transport plan $\sigma^*$ we can also define $\delta$-*close* or $\delta$-*approximate* transport plan as a transport plan $\sigma$ with cost $w(\sigma) \leq w(\sigma^*) + \delta$ that transports all of the mass from supply vertices to demand vertices.

The assignment problem is a special case of the optimal transport problem where the sets $A$ and $B$ each contain $n$ points, and every point in $A$ (resp. $B$) has a demand of $1/n$ (resp. supply of $1/n$).

## 1.2 Related Work

The problems of bipartite matching and optimal transport have been extensively studied due to their applications in combinatorial optimization, machine learning, and economics. Below, we summarize key contributions in this area.

### 1.2.1 Classical Matching Algorithms

The Hungarian algorithm provides a foundational approach for solving the min-cost perfect matching problem on bipartite graphs. By iteratively refining dual weights and augmenting along paths, it achieves a runtime of $\mathcal{O}(n^3)$ for graphs with $n$ vertices. Building upon this, Gabow's algorithm introduces the use of multiple augmenting paths per iteration, leveraging the Hopcroft-Karp algorithm to reduce runtime. This innovation achieves an asymptotic

complexity of $\mathcal{O}(\sqrt{n} \cdot m)$, where $m$ is the number of edges, but requires integer edge costs in the input graph.

Gabow and Tarjan's scaling algorithm further optimizes bipartite matching by introducing bit scaling for integer edge costs. The algorithm operates over $\mathcal{O}(\log nC)$ scales, such that $C$ is the maximum cost, achieving a runtime $\mathcal{O}(n^{2.5} \log nC)$. However, this approach assumes perfect matchings in input graphs and requires careful scaling of costs and demands, limiting its practical application in some scenarios.

### 1.2.2 Optimal Transport and Approximation Algorithms

There are many cases of the transportation problem that have been extensively studied. When all demands and supplies are positive integers, the problem is referred to as the Hitchcock-Koopmans transportation problem. If each demand and supply is equal to one, the problem reduces to the mininmum cost perfect (or maximum cardinality) matching problem. When the transportation cost between vertices satisfies a metric, the OT cost is also known as the Earth Mover's Distance. Furthermore, transportation cost is the $k$-th power of a metric with OT cost corresponds to the $k$-Wasserstein distance. These cases are well studied in the realm of machine learning. [1, 17, 19, 24, 26, 29] and have numerous applications across various domains [4, 6, 7, 9, 12, 28].

However, computing exact solutions to the optimal transport problem is computationally prohibitive due to its high complexity. As a result, extensive research has focused on developing efficient approximation algorithms.

Existing combinatorial algorithms for optimal transport include classical methods such as the Hungarian algorithm [18] for assignment problems, which operates in $\mathcal{O}(n^3)$ time, and the Gabow-Tarjan [14] scaling approach, with a complexity of $\mathcal{O}(n^{2.5} \log(nC))$ for balanced

transportation problems.

The most notable additive approximation algorithm build on these advancements by introducing a novel adaptation of the Gabow-Tarjan framework [20] and computes a $\delta$-close OT in running time of $\mathcal{O}\left(n^2\left(\frac{C}{\delta}\right) + n\left(\frac{C}{\delta}\right)^2\right)$.

### 1.2.3 Learning-Augmented Algorithms

The advent of machine learning inspires the philosophy of learning augmented algorithms that aims to utilize predictions to improve the performance of classic algorithms. By leveraging predictions to reduce uncertainty about the future, these methods often achieve better choices and improved competitive ratios. Notable examples include work on caching by Lykouris and Vassilvitskii [23], Rohatgi [27], and Jiang et al. [16]; on the classic secretary problem by Antoniadis et al. [3] and Dütting et al. [11]; on scheduling by Purohit et al. [25] and Lattanzi et al. [22]; on ski rental by Purohit et al. [25] and Anand et al. [2]; and on set cover by Bamas et al. [5].

With regards to minimum weight maximum cardinality bipartite matching, Chen et al. [8] proposed an approach that uses predicted dual weights as warm starts, significantly reducing the number of iterations required to compute optimal matchings. Their algorithm achieves a runtime of $\mathcal{O}(m\sqrt{n} + (m + n\log n)\|y^* - \hat{y}\|_0)$, where $\|y^* - \hat{y}\|_0$ is the sparsity of the difference between predicted and optimal dual weights and $m$ is the number of edges. Chen et al. learns their dual weights using a learning algorithm defined in Dinitz et al. [10]

## 1.3 Main Results

The goal of this work is to develop innovative techniques leveraging warm starts to devise asymptotically faster algorithms for minimum cost maximum cardinality matching and optimal transport problems. Two warm start algorithms are proposed, both utilizing dual weights as warm starts to compute a minimum-cost maximum cardinality matching. Additionally, one of these algorithms extends its utility to finding an optimal transport plan.

**$LMR$ algorithm with Warm Starts:** This algorithm introduces a novel scaling approach that leverages dual weights obtained from simplified versions of the transport or matching problem to significantly enhance computational efficiency. Specifically, dual weights are derived through multiple executions of an existing $\delta$-additive approximate optimal transport algorithm , known as the $LMR$ algorithm, [20] for varying values of $\delta$, gradually improving the precision until the desired $\delta$-error is achieved. These warm starts enable the computation of approximate matchings or transport plans with extremely small additive error $\delta$ significantly faster than the original $LMR$ algorithm which currently holds the state-of-the-art execution time for approximating the optimal transport plan, a generalization of bipartite matching, in the sequential setting.

The key improvement lies in the reduction of computational overhead by bounding the number of augmenting paths per scale to computationally manageable number. Therefore this allows the warm start algorithm to achieve a smoother trade-off between exact and approximate solutions, improving both efficiency and precision.
By incorporating warm starts, the algorithm achieves the following running times:

- For bipartite matching, the algorithm achieves an additive approximation with error $\delta$ in time:

$$\mathcal{O}\left(n^2 \cdot \min\left(\frac{C}{\delta}, \sqrt{n}\right)\right),$$

where $C$ is the largest cost in the given matching problem.

- For optimal transport, the algorithm achieves an additive approximation with error $\delta$ in time:

$$\mathcal{O}\left(n^2 \cdot \min\left(\sqrt{\frac{nC}{\delta}}, \frac{C}{\delta}\right)\right).$$

**Learning-Augmented Matching Algorithm:** The second algorithm employs a learning-augmented framework that utilizes machine-predicted dual weights, as described by Dinitz et al. [10]. By incorporating predicted dual weights into the initialization, the algorithm achieves greater efficiency than existing methods. The key improvement arises from augmenting along multiple augmenting paths per iteration, leveraging the Hopcroft-Karp technique to reduce the total number of iterations required.

This method achieves a faster runtime bound compared to prior learning-augmented techniques, with a time complexity of:

$$\mathcal{O}\left(\sqrt[4]{||y^* - \hat{y}||_1} \cdot n^{1/2} m\right),$$

# Chapter 2

# Preliminaries

In this section, we describe in detail four fundamental matching algorithms that form the foundation of our warm start approaches. We begin in the unweighted setting with a description of the Hopcroft-Karp algorithm for maximum cardinality matching in $\mathcal{O}(n^{2.5})$ by computing multiple augmenting paths per iteration. Computing augmenting paths is a fundamental approach for constructing maximum cardinality matching for these network flow algorithms. Next, in the weighted setting we describe the Hungarian Algorithm which is a fundamental algorithm for computing minimum cost maximum cardinality matching. Finally we describe Gabow's algorithm and Gabow and Tarjan's algorithm which are examples of scaling algorithms that can be seen as extensions of the Hopcroft-Karp algorithm to the weighted setting.

Throughout this chapter we denote $B_F$ as the set of free $B$ vertices and $A_F$ as the set of free $A$ vertices. Given a bipartite graph $G$ and matching $M$ we can construct a directed residual graph $\overrightarrow{G}$ that captures the set of alternating paths. This is accomplished by assigning each edge of $E$ a direction. Any edge in $M$ is directed from $A$ to $B$, and any edge not in $M$ is directed from $B$ to $A$. Any directed path in $\overrightarrow{G}$ is an alternating path in G with respect to M. Any path $P$ in this graph $\overrightarrow{G}$ that starts with a free $b \in B_F$ and ends with free $a \in A_F$ forms an augmenting path. Augmentation is the process of creating a new matching $M'$ by making all the non matching edges matching edges and vice versa. Augmenting along these paths is critical, as it allows the algorithm to obtain a new matching $M'$ with one additional

edge. This augmentation process is the same as taking the symmetric difference between $M$ and $P$, denoted $M' = M \oplus P$. This symmetric difference results in $M'$ with $|M'| = |M| + 1$. This concept of a residual graph is used throughout the algorithms described in this chapter.

## 2.1 Hopcroft-Karp Algorithm

Each iteration of the Hopcroft-Karp algorithm is referred to as a phase. Each phase uses BFS step and DFS step to find a set of augmenting paths, followed by augmentation along all paths in this set.

1. **BFS Step:** The algorithm starts by making a residual graph $\overrightarrow{G}$ w.r.t the current matching $M$. A breadth-first search (BFS) is executed from all free vertices in $B_F$ in order to find the shortest augmenting path.

2. **DFS Step:** DFS is utilized to determine the set of "vertex-disjoint augmenting paths" $P$ in the admissible graph. At the end of this step no more such vertex disjoint paths can be found. This is accomplished by conducting partial depth-first searches (DFS), each starting from a free vertex in $B$. For each free vertex, the algorithm initiates a DFS from $v$. All shortest augmenting path (identified in BFS step) $P$ found is added to the set $S$, and the corresponding partial DFS is terminated. If no augmenting path is discovered, the partial DFS concludes upon backtracking from $v$.

Augmentation along these set of paths $P \in S$ is viable since these paths are disjoint therefore the final matching is valid.

## 2.2 Hungarian Algorithm

The Hungarian Algorithm is used [18] to find the optimal perfect matching (matching with least cumulative cost and maximum size) on a bipartite graph $G(A \cup B)$. Every edge $(a, b) \in A \times B$ has an associated non-negative cost $c(a, b)$ allowing us to define the cost of matching as

$$c(M) = \sum_{(a,b)\in M} c(a, b)$$

We aim to find a maximum cardinality matching $M$ that minimizes the overall matching cost $c(M)$. The algorithm operates within a primal-dual framework, maintaining a set of dual weights for each vertex and adjusting them iteratively to preserve feasibility. The Hungarian algorithm constructs a maximum cardinality matching while maintaining dual feasibility conditions which ensures minimizes the cost of matching.

### 2.2.1 Dual Feasibility

Throughout the Hungarian algorithm a set of dual weights and a matching is maintained. To ensure a minimum cost matching we introduce these feasibility conditions on the dual weights. A matching $M$ and set of dual weights $y(\cdot)$ are feasible if for every edge $(a, b)$,

$$y(a) + y(b) \leq c(a, b) \quad \text{if } (a, b) \notin M. \tag{2.1}$$

$$y(a) + y(b) = c(a, b) \quad \text{if } (a, b) \in M. \tag{2.2}$$

Using the dual weights of vertices $a$ and $b$ and the edge cost $c(a, b)$, slack on edge $(a, b)$ is defined as follows:

$$\mathrm{s}(a, b) = c(a, b) - (y(a) + y(b))$$

From the definition and feasibility conditions, any edge in the matching set $M$ has zero slack, while edges not in $M$ have non-negative slack. Admissible edges are edges that satisfy conditions (2.1) and (2.2). Throughout the algorithm these edges for all intermediate matchings do not violate feasibility. The Hungarian Algorithm adds these admissible edges by augmenting along augmenting paths. In the Hungarian algorithm, finding and augmenting along an augmenting path of admissible edges in each iteration is sufficient to progressively build maximum cardinality matching. With the help of Lemma 2.1 below we can see that building a feasible maximum cardinality matching will ensure a minimum cost maximum cardinality matching.

**Lemma 2.1.** *A maximum cardinality matching $M$ that is feasible with respect to the $y(\cdot)$ on $A \cup B$ is a minimum cost matching.*

*Proof.* Let $\mathbf{M}$ and a duals weights $y(\cdot)$ be feasible as defined in (2.2). The cost of the matching is as follows.

$$c(\mathbf{M}) = \sum_{(a,b)\in M} c(a,b)$$

Hence, according to the dual feasibility conditions.

$$\sum_{(a,b)\in \mathbf{M}} y(a) + y(b) = c(\mathbf{M})$$

On the other hand, consider another maximum cardinality matching $M$ such that $M \oplus \mathbf{M} \neq \emptyset$. Then cost of the matching $c(M)$ is defined as follows.

$$c(M) = \sum_{(a,b)\in M} c(a,b)$$

Now, according to the dual feasibility conditions defined in (2.1).

$$\sum_{(a,b)\in M} y(a) + y(b) \leq c(M)$$

Therefore for all perfect matching M, $c(\mathbf{M}) \leq c(M)$ hence $\mathbf{M}$ is minimum cost matching. □

### 2.2.2 Algorithm

The Hungarian algorithm due to Lemma 2.1 maintains feasible dual weights and increases the cardinality of its matching by 1 in every iteration. Initially every vertex has a dual weight of 0, therefore the feasibility condition (2.1) holds. Note all edges have a non-negative cost. The algorithm runs in $\mathcal{O}(n)$ iterations since there are $n$ points to match while maintaining the feasibility of $M, y(\cdot)$. Each iteration takes $\mathcal{O}(n^2)$ time, giving a total time of $\mathcal{O}(n^3)$ for the algorithm. A single iteration of the Hungarian algorithm is known as one Hungarian Search.

**Hungarian Search**: To find the augmenting paths, the algorithm performs dual adjustments using a procedure known as the 'Hungarian search.' First, a residual graph $\overrightarrow{G}_M$ is created by orienting non-matching edges from $B$ to $A$; matching edges from $A$ to $B$. Each edge in the residual graph is assigned a cost equal to the slack of the edge, as previously defined. Additionally, a source vertex $s$ is introduced, with zero-cost edges connecting $s$ to each free vertex in $B$. The algorithm proceeds by running Dijkstra's algorithm from $s$ to compute the shortest distance from $s$ to each vertex in the graph, denoted as $\ell_v$. The minimum distance from $s$ to a free vertex in $A$ is represented as $\ell$. If $\ell = \infty$, no augmenting path exists, implying that the current matching is optimal. Otherwise, using these shortest-path

distances, the algorithm updates the dual values as follows: for vertices $v \in B$, if $\ell_v \geq \ell$, $y(v)$ stays the same; otherwise, if $\ell_v < \ell$, $y(v)$ is updated as $y(v) \leftarrow y(v) + (\ell - \ell_v)$. Similarly, for vertices $v \in A$, if $\ell_v < \ell$, $y(v)$ is updated as $y(v) \leftarrow y(v) - (\ell - \ell_v)$.

### 2.2.3 Invariants

These are the invariants of the Hungarian algorithm. Suppose $M$ represent a feasible matching and $y(\cdot)$ denote the set of dual weights maintained by the algorithm before the start of a Hungarian Search. After the Hungarian search is finished let $y'(\cdot)$ represent the updated dual weights. [18]:

1. For all non-matching edge, the condition (2.1) remains satisfied with respect to $y'(\cdot)$

2. For all matching edge, condition (2.2) holds with respect to $y'(\cdot)$ .

## 2.3 Scaling Algorithms

Both of the following algorithms assumes that the bipartite graph $G$ has a perfect matching and all of the edge costs are integers. Each iteration of both of these algorithms is known as a *scale* where the algorithm for each scale takes as input a balanced bipartite graph $G$ with a optimal matching $M^*$ whose cost is $\mathcal{O}(n)$. The algorithm for each scale returns a maximum-cardinality matching and a set of dual weights. Each scale executes multiple iterations where each iteration is known as a *phase*.

### 2.3.1 Gabow's Algorithm

Gabow's Algorithm [13] iteratively refines a matching on a bipartite graph $G$ to achieve a minimum cost perfect matching. Unlike the Hungarian algorithm that augments one augmenting path at a time Gabow's Algorithm augments all of the possible augmenting paths found at the end of one iteration with the help of the Hopcroft-Karp algorithm allowing it to be $\sqrt[4]{n}$ faster than Hungarian Algorithm.

#### Scaling Paradigm

In order to bound the cost of matching to $\mathcal{O}(n)$ for each scale Gabow's algorithm employs a cost scaling routine. Initially, the cost of each edge $(a, b)$ is repeatedly halved using $\lfloor \frac{c(a,b)}{2} \rfloor$ until the largest edge cost of $G$ is reduced to 1, which defines the first scale. In subsequent scales, the edge costs are then doubled every time.

In each scale, the Gabow's algorithm computes a feasible perfect matching $M_i$ and a set of dual weights $y_i(\cdot)$. For any edge $(u, v)$, let $c_i(u, v)$ be the cost of the edge during the $i^{\text{th}}$ scale. The dual weights from scale $i$ are then used to generate an initial set of feasible dual weights for scale $i + 1$. This is done by setting, for every $v \in A \cup B$, $y_{i+1}(v) \leftarrow 2y_i(v) - 1$. In order to make sure that the matching found in scale $i + 1$ has a cost of $\mathcal{O}(n)$ we calculate reduced edge costs.

The reduced cost of any edge $(u, v)$ is defined as:

$$c'_{i+1}(u, v) = c_{i+1}(u, v) - y_{i+1}(u) - y_{i+1}(v).$$

Since matching $M_i$ is feasible w.r.t $y_i(\cdot)$, the reduced costs $c_{i+1}(\cdot, \cdot)$ has a cost of $\mathcal{O}(n)$. The algorithm then computes a feasible matching for the reduced costs $c'_{i+1}(\cdot, \cdot)$, using an $\mathcal{O}(n^{2.5})$

time algorithm.

The resulting dual weights at the end of the $\mathcal{O}(n^{2.5})$ algorithm are then added to the original scaled dual weights at the start of scale $i+1$ in order to remain feasible w.r.t to the original costs $c_{i+1}(\cdot)$ for scale $i$.

**Algorithm for a Single scale:**

This section describes the algorithm for computing a feasible matching for each scale. Using the scaling paradigm, we can assume that the optimal matching $M^*$ has cost $c(M^*) = \mathcal{O}(n)$, and each edge has a cost of at least $-1$. Initially, the algorithm assigns every vertex a dual weight of 0; since all edges are initially not in the matching, this initial dual assignment is feasible. Each phase can be described as follows:

**Step 1** Hungarian Search: A Hungarian Search is conducted (2.2) to locate an augmenting path in $G$. This search helps identify edges along which adjustments can improve the current matching. During this search a residual subgraph (referred to in section 2.2) $G' = (V, E')$, where $E' = \{(u, v) \mid s(u, v) = 0\}$ is constructed. This subgraph contains only the edges with zero slack, and thus are eligible for inclusion in a matching.

**Step 2** Hopcroft-Karp: The Hopcroft-Karp [15] is utilized to compute a maximum cardinality matching on $G'$. This step expands the current matching $M$ as much as possible within the feasible set of edges in $G'$.

The algorithm repeats these steps until $M$ becomes a perfect matching on $G$. By iteratively finding augmenting paths and updating the matching, Gabow's Algorithm ensures that the resulting matching is optimal in terms of cardinality and cost.

### 2.3.2 Gabow and Tarjan's Algorithm

The Gabow-Tarjan (GT) scaling algorithm [14] also solves the min-cost matching problem on bipartite graphs. This algorithm operates over $\mathcal{O}(\log nC)$ scales. Each scale of this algorithm takes a bipartite graph, each partite set having $n$ vertices. The only assumption here for this algorithm is that the optimal matching $M^*$ should have a cost of $\mathcal{O}(n)$.

Each scale takes $\mathcal{O}(\sqrt{n})$ phases, with each phase taking $\mathcal{O}(n^2)$ time. Consequently, the overall running time of $\mathcal{O}(n^{2.5} \log nC)$. Note for the sake of this thesis we will assume that there exists a perfect matching even though reductions for otherwise are available in [14].

#### 1 - Feasibility

The Gabow-Tarjan (GT) algorithm uses modified feasibility conditions as compared to the Hungarian Algorithm. A set of dual weights $y(\cdot)$ are maintained throughout the execution of the algorithm. A set of dual weights $y(\cdot)$ are considered 1-feasible with respect to a matching $M$ if, for every edge $(u, v)$,

$$y(u) + y(v) \leq c(u, v) + 1 \quad \text{if } (u, v) \notin M,$$

$$y(u) + y(v) = c(u, v) \quad \text{if } (u, v) \in M.$$

This relaxation of the feasibility conditions allows us to bound the running time of this algorithm. It also leads to a error of $+n$ at the very most

### Scaling Paradigm

The ultimate objective of the Gabow-Tarjan algorithm is to obtain an exact optimal matching. This exactness is accomplished through scaling. Initially, all edge costs are multiplied by $n+1$, ensuring that difference between optimal matching and the next best matching is greater than or equal to $n+1$. This allows all 1-feasible perfect matching with scaled costs to be optimal. Under these scaled costs, the maximum edge cost becomes $\mathcal{O}(nC)$.

In order to bound the number of phases to $\mathcal{O}(\sqrt{n})$, the Gabow-Tarjan algorithm needs the optimal matching to have a cost of $\mathcal{O}(n)$. This assumption is met by using a bit-scaling approach, executing $\mathcal{O}(\log nC)$ scales. In scale $i$, $i$ most significant bits of each edge cost are used. Therefore in the first scale the the optimal matching is $\mathcal{O}(n)$.

The mechanics of the scaling paradigm employed by the Gabow and Tarjan's algorithm is exactly same as the scaling paradigm described in Gabow's algorithm 2.3.1 except for the fact that at the end of every scale a 1-feasible matching and dual weights are produced.

### Algorithm for a Scale

At the beginning of the algorithm the dual weight of every vertex is 0 since the matching set is empty. Each scale takes $\mathcal{O}(\sqrt{n})$ phases, with each phase containing two stages.

**Hungarian Search:** The first stage is the Hungarian search. This step closely resembles the procedure described in Section 2.2, Since the feasibility condition is different from the Hungarian Algorithm the slack for Gabow and Tarjan's is defined as follows:

$$s(u,v) = c(u,v) + 1 - y(u) - y(v) \quad \text{if } (u,v) \notin M,$$

$$s(u,v) = y(u) + y(v) - c(u,v) = 0 \quad \text{if } (u,v) \in M.$$

With this adjusted definition of slack, the GT algorithm performs the Hungarian Search as described in section 2.2.2 to change the dual weights. These dual adjustments do not violate 1-feasibility.

**Partial DFS:** In the second stage, the algorithm performs the exact same partial DFS as the DFS step from Hopcroft-Karp's Algorithm in section 2.1 from each free vertex of $B$ in the admissible graph.

Note non-matching edges that enter after augmenting along some augmenting path $P$ could now be infeasible due to the $+1$ relaxation of non-matching edges. To maintain feasibility during augmentation the dual weight of $b \in B$ on $P$ decreases by 1.

The algorithm for a single scale is repeatedly executed until a perfect matching is obtained.

### Running Time of Algorithm

For a single scale the GT algorithm is bounded by $\mathcal{O}(n^{2.5})$. Therefore, for all $\mathcal{O}(\log nC)$ scales, runtime complexity is $\mathcal{O}(n^{2.5} \log nC)$. Each phase consists of two steps as described above.

For the first step, the Hungarian search executes a Dijkstra's algorithm. Since optimal matching is $\mathcal{O}(n)$, Dijkstra's algorithm can be shown to run in $\mathcal{O}(n^2)$ time using special data structures. The second step performs a DFS which also takes $\mathcal{O}(n^2)$ time. Thus, each phase of the GT algorithm is $\mathcal{O}(n^2)$.

Finally, the total number of phases is bounded because the optimal matching cost in every scale is $\mathcal{O}(n)$. Then at any point in the algorithm, for a phase, $y_{\max}|B_F| = \mathcal{O}(n)$ where $y_{\max}$ is the largest dual weight and $B_F$ is the number of free B vertices at that point in the algorithm. Due to this property we can easily observe $y_{\max}$ increases by at least one every phase (except possibly the first) and $|B_F|$ decreases by at least one every phase until the

matching is perfect. Therefore after $\mathcal{O}(\sqrt{n})$ phases $y_{\max}$ is at least $\mathcal{O}(\sqrt{n})$ and there are atmost $\mathcal{O}(\sqrt{n})$ free B points left. Therefore the remaining vertices are matched in $\mathcal{O}(\sqrt{n})$ phases implying the total number of phases is $\mathcal{O}(\sqrt{n})$.

# Chapter 3

# Scaling Algorithm with Warm Starts

In this section, we will explore the first warm-start algorithm in detail. Before delving into its specific workings, we will first review the $LMR$ algorithm, a foundational component of the warm-start method.

## 3.1 $LMR$ Algorithm

### 3.1.1 Introduction

$LMR$ algorithm is a deterministic primal-dual algorithm presented from [20] to compute a $\delta$-close solution in $\mathcal{O}\left(\frac{n^2C}{\delta} + \left(\frac{nC}{\delta}\right)^2\right)$ time where $C$ is the largest edge cost and $n$ is the number of supply and demand points. The first term in the running time is possible if $\frac{C}{\delta} = \mathcal{O}(n)$. Note that $\frac{C}{\delta}$ is known as the diameter.

The problem is transformed to one with integer demands and supplies in $\mathcal{O}(n^2)$ time. Given these transformed demands and supplies, the algorithm scales down the cost of each edge $(a, b)$ to $\left\lfloor \frac{2c(a,b)}{(1-\epsilon)\delta} \right\rfloor$ for small constant $\epsilon$ $(0 < \epsilon < 1)$. The algorithm runs in at most $\left\lfloor \frac{2C}{(1-\epsilon)\delta} \right\rfloor + 1$ phases. Each phase comprises two steps:

1. **Hungarian Search:** This mechanics of this algorithm is the exact same as the Hungarian Search step described in section 2.2.2. The goal is to locate an augmenting path

of zero-slack edges.

2. **Partial DFS:** In this step, a DFS is initiated from each free supply vertex to find admissible augmenting paths. Supply is transported on along these zero slack augmenting paths

After a 1-feasible plan is formed a maximum feasible plan with additive error $\delta$ can then be constructed.

### Definitions and Notations

A vertex $a \in A$ with demand $d_a$ (or $b \in B$ with supply $s_b$) is called *free* w.r.t a transportation plan if

$$d_a - \sum_{b \in B} \sigma(a, b) > 0,$$

and similarly, $b$ is free if

$$s_b - \sum_{a \in A} \sigma(a, b) > 0.$$

Please look at the Important Concepts section 1.1 to find any unfamiliar definitions or notations.

At any point in the algorithm, $A_F$ and $B_F$ represent the sets of unsaturated demand and supply nodes, respectively.

In order to scale the original costs to integral costs for all edges $(a, b)$ let $c(a, b) = \left\lfloor \frac{2c(a,b)}{\delta(1-\varepsilon)} \right\rfloor$. Note epsilon is a small number and in the asymptotic analysis not significant and in some instance can be limited for brevity. The cost $w(\sigma)$ of any transport plan $\sigma$ is defined w.r.t the scaled cost $c(\cdot, \cdot)$.

Given transport plan $\sigma$ and a dual weight $y(v)$ for each vertex, the pair $\sigma, y(\cdot)$ is defined as

*1*-feasible if, for any two supply or demand vertices, the following conditions hold:

$$y(a) + y(b) \leq c(a,b) + 1 \quad \text{if } \sigma(a,b) < \min\{s_b, d_a\}, \tag{3.1}$$

$$y(a) + y(b) \geq c(a,b) \quad \text{if } \sigma(a,b) > 0. \tag{3.2}$$

These conditions are adapted from those in Gabow and Tarjan's algorithm, but they apply here to costs scaled by $\frac{2}{\delta}$ and rounded down. This algorithm maintains a transport plan with 1-feasible dual weights throughout the algorithm. A transport plan satisfying these conditions and being maximum is referred to as a *1-optimal transport plan.* Unlike GT algorithm, which is formulated for equal total supply and equal total demand transportation problems (where a max transport plan also satisfies every demand constraint), this algorithm may encounter unsatisfied demands. To address this, an additional condition is introduced for any 1-optimal transport plan $\sigma$:

(A) All demand vertices $a \in A$ have a dual weight $y(a) \leq 0$, and if $y(a) = 0$ then $a$ is a free or unsaturated demand vertex.

**Residual Graph Construction**

Every 1-feasible transport plan $\sigma$ has a corresponding constructed directed residual graph denoted $\overrightarrow{G}_\sigma$, and constructed as follows. For each edge $(a,b) \in A \times B$:

1. If $\sigma(a,b) = 0$, a forward edge oriented $b$ to $a$ is added with remaining or "residual" capacity set to the minimum of the demand of a and supply of b.

2. If $\sigma(a,b)$ equal to the minimum of the demand of a and supply of b, a backward edge from $a$ to $b$ is added with remaining $\sigma(a,b)$.

3. If $0 < \sigma(a,b) < \min\{d_a, s_b\}$, both forward and backward edge with residual capacity $\min\{d_a, s_b\} - \sigma(a,b)$ is added.

The Both forward and backward edge from $(a,b)$ in $\overrightarrow{G}_\sigma$, have $c(a,b) = \left\lfloor \frac{2c(a,b)}{\delta} \right\rfloor$.

**Augmenting Paths and Flow Adjustment**

Any directed path in the residual graph $\overrightarrow{G}_\sigma$ that starts from an unsaturated supply vertex and ends at an unsaturated demand vertex is known as an *augmenting path*. An augmenting path alternates between forward and backward edges, with both the starting and ending edges being forward edges. Given an augmenting path $P$, the supplies transported along $P$ can be increased by $k \geq 1$ units as follows:

1. For each forward edge $(a,b)$ in $P$, add k units to $\sigma(a,b)$.

2. For each backward edge $(a,b)$ in $P$, remove k units from $\sigma(a,b)$.

**Slack and Admissible Graph:**

Slack on an edge is defined as follows:

$$s(a,b) = c(a,b) + 1 - y(a) - y(b) \quad \text{if } (a,b) \text{ is a forward edge,} \tag{3.3}$$

$$s(a,b) = y(a) + y(b) - c(a,b) \quad \text{if } (a,b) \text{ is a backward edge.} \tag{3.4}$$

An edge $(a,b)$ in $\overrightarrow{G}_\sigma$ is called *admissible* if $s(a,b) = 0$. The *admissible graph* $\overrightarrow{A}_\sigma$ is the subgraph of $\overrightarrow{G}_\sigma$ consisting of all admissible edges.

### 3.1.2 Scaling Supplies and Demands

The real demands and supplies of the original Optimal Transport (OT) problem are transformed to integer values, enabling the use of the traditional augmenting path framework to approximate the transformed problem within $O\left(\frac{n^2C}{\delta} + \frac{nC^2}{\delta^2}\right)$ time. The integral transport plan is mapped back to a feasible transport plan for the initial real-valued demands and supplies, with a total accuracy loss in cost of at most $\epsilon U\delta$.

Given a small positive $\varepsilon < 1$ define $\alpha = \frac{2nC}{\epsilon U\delta}$. Given an input $I$ to a transportation problem, where each vertex $a \in A$ has demand $d_a$ and vertex $b \in B$ has a supply $s_b$. We form a new instance $I'$ by upscaling the demands and supplies as follows:

$$d'_a = \lceil d_a\alpha \rceil, \quad s'_b = \lfloor s_b\alpha \rfloor.$$

Let the total supply be $D = \sum_{a\in A} d_A$. Since total demands are scaled by $\alpha$ and rounded up, we have:

$$\mathcal{D} = \sum_{a\in A} d'_a = \sum_{a\in A} \lceil d_a\alpha \rceil \geq \alpha \sum_{a\in A} d_A = \alpha D. \tag{3.5}$$

Similarly, since supplies are rounded down total scaled supply is

$$\mathcal{U} \leq \alpha U. \tag{3.6}$$

Therefore from (3.6) and (3.5) the total supply $\mathcal{U}$ in $I'$ is still less than the total demand $\mathcal{D}$.

### 3.1.3 Algorithm for Scaled Supplies, Demands and Costs

Initially, the transport plan $\sigma$ is set to zero. The dual weights of each vertex are also initialized to zero. Together, $\sigma$ and $y(\cdot)$ form a 1-feasible OT plan. The algorithm operates in phases and stops once $\sigma$ becomes a maximum transport plan. Each phase consists of 2 main stages: a Hungarian Search and the computation and augmentation of admissible paths.

#### Hungarian Search

In this step, the algorithm adjusts dual weights to ensure at least one augmenting path of admissible edges. This step is extremely similar to Hungarian Search defined in section 2.2.2. Let the residual network be denoted as $G_\sigma$. Similar to the Hungarian Search defined previously Dijkstra's is executed at $s$ in $G_\sigma$ to compute the shortest path $\ell_v$ from $s$ to each vertex $v \in A \cup B$. A dual weight adjustment is then applied as follows:

- For each vertex $v \in A \cup B$, if $\ell_v \geq \ell_t$ (where $\ell_t$ is the shortest path to $t$), the dual weight $y(v)$ remains unchanged.

- Otherwise:

$$\text{If } v \in A, \text{ set } y(v) \leftarrow y(v) - \ell_t + \ell_v, \tag{3.7}$$

$$\text{If } v \in B, \text{ set } y(v) \leftarrow y(v) + \ell_t - \ell_v. \tag{3.8}$$

These dual updates ensure the transport plan is 1-feasible with one admissible path in the residual network.

### Partial DFS

Let $X$ denote the set of free supply nodes in $B$. Like the partial DFS step in GT algorithm 2.3.2 or Hopcorft Carp algorithm 2.1 a DFS from each node in $X$ is initiated.

During the DFS from a free supply vertex $b \in X$, if an unsaturated demand vertex is encountered, an augmenting path $P$ is identified. DFS then ends, retaining only edges of $P$ and deleting all other edges visited in the DFS and accordingly updating $X$ based on the augment procedure (removing vertices once they are no longer free). This step ends when $X$ becomes empty. Overall this step is extremely similar to the Hopcroft-Karp DFS step in section 2.1

### Augment Procedure

The bottleneck capacity $r_P$ of $P$ is defined as the minimum residual supply $b$, the residual demand $a$, and the smallest residual capacity in augmenting path $P$ (b($P$)):

$$r_P = \min \left\{ s_b - \sum_{a' \in A} \sigma(a', b),\ d_a - \sum_{b' \in B} \sigma(a, b'),\ \mathrm{b}(P) \right\}.$$

The transport plan $\sigma$ is augmented along $P$ by updating $\sigma$ depending on the following edge:

- **Forward edge**: $(a', b')$ on $P$, add $r_P$ to $\sigma(a', b')$.
- **Backward edge**: $(a', b')$ on $P$, subtract $r_P$ from $\sigma(a', b')$.

After augmentation, we have a new transport plan.

The invariants of this algorithm are the following

1. The algorithm maintains a 1-feasible transport plan due to the Hungarian Algorithm

(refer Theorem 1 and 2 to see the )

2. In each phase, one augmenting path is found and at the end of the partial DFS, there are zero augmenting paths with 1-feasible edges left

### 3.1.4 Constructing Transport Plan for Original Capacities

Let $\sigma'$ be a max 1-feasible plan for the scaled transportation problem $I'$. Then we can define a transport plan $\sigma$ for transportation plan $I$ defined as $\forall(a,b), \quad \sigma(a,b) = \frac{\sigma'(a,b)}{\alpha}$. Note that $\sigma$ may not be feasible or maximum for $I$:

1. Non-Maximality: $\sigma$ may not fully use the available supply at some nodes $b \in B$, as

$$\sum_{a \in A} \sigma(a,b) = \sum_{a \in A} \frac{\sigma'(a,b)}{\alpha} = \frac{s'_b}{\alpha} \leq s_b.$$

2. Infeasibility: $\sigma$ may not meet all demands at some nodes $a \in A$, as

$$\sum_{b \in B} \sigma(a,b) = \sum_{b \in B} \frac{\sigma'(a,b)}{\alpha} = \frac{d'_a}{\alpha} \geq d_a.$$

The transport plan cost given by $w(\sigma) = \frac{w(\sigma')}{\alpha}$. Converting $\sigma$ to max 1-feasible transport plan for $I$ is achieved due to the following procedures:

1. Feasibility Adjustment: To address excess supply $\kappa_a$ at each demand node $a \in A$, iteratively pick an edge $(a,b)$, and reduce $\sigma(a,b)$ and $\kappa_a$ by $\min(\kappa_a, \sigma(a,b))$. Repeat this process until $\kappa_a = 0$ for all $a \in A$. The total remaining supply in $\sigma$ is at most $\frac{2n}{\alpha}$. The cost $w(\sigma) \leq \frac{w(\sigma')}{\alpha}$ remains unaffected.

2. Maximality Adjustment: To convert $\sigma$ to a maximum transport plan, assign the remaining $\frac{2n}{\alpha}$ supplies to leftover demands. The maximum cost to transport one unit of supply is $C$. This adjustment increases the cost by at most $\frac{2nC}{\alpha}$, so

$$w(\sigma) \leq \frac{w(\sigma')}{\alpha} + \frac{2nC}{\alpha} \leq \frac{w(\sigma')}{\alpha} + \epsilon U \delta. \tag{3.9}$$

### 3.1.5 Proof of Correctness

These correctness proofs and in the next subsection the efficiency proofs are taken from [20] and are added as foundation to explain warm start algorithm. These proofs come from the original paper and are building blocks for the warm start proofs. The notation used in these proofs will be borrowed for the warm start proofs.

**Lemma 3.1.** *Let $\sigma^*$ be the optimal transport plan for original costs, supplies and demands and let $\sigma'_{\text{OPT}}$ be the optimal transport plan for the scaled version. Then $w(\sigma'_{\text{OPT}}) \leq \alpha w(\sigma^*)$.*

*Proof.* Let's define a transport plan $\sigma'(a, b) = \alpha\sigma^*(a, b)$ which is initially invalid because the total supply from any $b \in B$ could exceed $\overline{s}_b$ by an amount:

$$\kappa_b = \sum_{a \in A} \sigma'(a, b) - \overline{s}_b = \sum_{a \in A} \alpha\sigma^*(a, b) - \overline{s}_b = \alpha s_b - \lfloor \alpha s_b \rfloor.$$

This is because $\sigma^*$ is a maximum transport plan (as defined in section 3.1.2), which implies $\sum_{a \in A} \sigma^*(a, b) = s_b$ eventhough $\sigma'$ does satisfy every vertex's demand. To make $\sigma'$ a valid transport plan, we iteratively adjust for each supply node $b \in B$ by choosing an edge $(a, b)$ and reducing both $\sigma'(a, b)$ and the remaining supply $\kappa_b$ by $\min\{\sigma'(a, b), \kappa_b\}$. This keeps happening till all supply constraints are met exactly and thus become a valid transport plan for $\mathcal{I}'$. This procedure is similar to procedure defined in section 3.1.2 Since the amount of

supply transported along any edge $(a, b)$ in $\sigma'$ is at most $\alpha\sigma^*(a, b)$, we have:

$$w(\sigma') = \sum_{(a,b)\in A\times B} \sigma'(a, b)c(a, b) \leq \sum_{(a,b)\in A\times B} \alpha\sigma^*(a, b)c(a, b) = \alpha w(\sigma^*).$$

□

**Lemma 3.2.** *Give a transport plan $\sigma$ and dual weights $y(\cdot)$ such that $\sigma$ is 1-optimal transport plan that satisfies (A) as defined in 3.1.1. Let $\sigma' = \sigma'_{\text{OPT}}$ be the optimal transport plan. Then, $w(\sigma) \leq w(\sigma') + \delta'\mathcal{U}$.*

*Proof.* The proof of this lemma is the same as the proof given in the appendix of [20]. The proof is omitted in this thesis as its detailed mechanics are tedious and do not contribute significantly to the discussion. □

**Theorem 3.3.** *If $\sigma^*$ is the optimal transport plan for the original instance $\mathcal{I}$, then the transport plan $\sigma$ produced by this algorithm is $w(\sigma) \leq w(\sigma^*) + U\delta$.*

*Proof.* Let $\sigma'_{\text{OPT}}$ be the optimal solution for scaled input instance (as described in 3.1.2 and 3.1.1) $\mathcal{I}'$. The result from Lemma 3.2 can be combined with the result in Lemma 3.1 to get $w(\sigma') \leq \alpha w(\sigma^*) + (1 - \varepsilon)\mathcal{U}\delta$. Finally by combining (3.6) and (3.9) in section 3.1.2 we get $w(\sigma) \leq w(\sigma^*) + (1 - \varepsilon)\mathcal{U}\delta/\alpha + \varepsilon U\delta \leq w(\sigma^*) + (1 - \varepsilon)U\delta + \varepsilon U\delta = w(\sigma^*) + U\delta$ □

### 3.1.6 Running Time Analysis

**Lemma 3.4.** *This algorithm executes at most $\left\lfloor \frac{2C}{\delta'} \right\rfloor + 1$ phases.*

*Proof.* In the beginning there are no augmenting paths due to the second invariant stated in section 3.1.3. Therefore any path from an unsaturated ($B$) supply vertex to an unsaturated

$(A)$ demand vertex will have a cost of at least 1. Let $b \in B$ be a free supply vertex at the beginning of any phase. Then $b$ is a free supply vertex every previous phase. Therefore due to equation (3.8) the dual weight of $b$ increases by at least one therefore it is sufficient to bound the maximum value of this $b$ to bound the number of phases.

Assume for the sake of contradiction, $b \in B$ has a dual weight $y(b) \geq \lfloor \frac{2C}{\delta'} \rfloor + 2$. Then for any free demand vertex $a \in A$ the dual weight $y(a) = 0$ or if there are no free demand vertices at the end of the algorithm then the last demand vertex $a \in A$ to be saturated at the end of the algorithm also has dual weight of 0. Then sum of dual weights of edge $(a, b)$ does not meet feasibility conditions leading to a contradiction. Therefore the maximum dual weight of $b$ is $\lfloor \frac{2C}{\delta'} \rfloor + 1$ and in turn the maximum number of phases is $\lfloor \frac{2C}{\delta'} \rfloor + 1$. □

**Lemma 3.5.** *Let $\mathbb{P}$ be all augmenting paths found at the end of this algorithm. Then $\sum_{P \in \mathbb{P}} |P| = \mathcal{O}(\frac{nC^2}{\varepsilon(1-\varepsilon)\delta^2})$; here $|P|$ is the total count of edges on $P$.*

*Proof.* To bound the length of all augmenting paths find the bound the net cost of all of the augmenting paths. Net cost of an augmenting path ($\phi(P)$) is defined as the difference between the total cost of the forward edges and the total cost of the backward edges.

$$\Phi(P) = \sum_{(a',b') \in P^{\uparrow}} (\overline{c}(a', b') + 1) - \sum_{(a',b') \in P^{\downarrow}} \overline{c}(a', b').$$

First, the net-cost of $P$ is upper bounded by $\lfloor 2C/\delta' \rfloor + 1$:

$$\begin{aligned} \Phi(P) &= \sum_{(a',b') \in P^{\uparrow}} (y(a') + y(b')) - \sum_{(a',b') \in P^{\downarrow}} (y(a') + y(b')) && (3.10) \\ &= y(b) \leq \lfloor 2C/\delta' \rfloor + 1. && (3.11) \end{aligned}$$

In an augmenting path all edges have a slack of zero so the first and second term of the net cost definition is equivalent to the corresponding dual weight sum as seen in (3.10) because

$P$ is an augmenting path in the admissible graph, where the slack on every edge of $P$ is zero. Equation (3.11) follows from Lemma 3.4. Let $\sigma$ be the transport plan when $P$ is discovered by the algorithm, and let $\sigma'$ be the transport plan obtained by augmenting $\sigma$ along $P$ by $r_P$ units. Now from the proof of theorem 2.4 in [20] we can get the following equation:

$$\sum_{P\in\mathbb{P}}(r_P\Phi(P)) = \overline{w}(\overline{\sigma}) + \sum_{P\in\mathbb{P}} r_P\lceil|P|/2\rceil.$$

$$\mathcal{U}\left(\lfloor 2C/\delta'\rfloor + 1\right) \geq \sum_{P\in\mathbb{P}}\lceil|P|/2\rceil. \tag{3.12}$$

Equation (3.12) follows from the facts that transport plan is non negative, bottleneck capacity is positive and scaled total supplies is less than $\alpha$ multiplied by original total supplies meaning the total flow is at most $\mathcal{U}$. The final result is:

$$\sum_{P\in\mathbb{P}}|P| = \mathcal{O}\left(\frac{nC}{\epsilon\delta}\cdot\frac{C}{(1-\epsilon)\delta}\right).$$

□

**Theorem 3.6.** *The total execution time of this algorithm is equal to* $\mathcal{O}(\frac{n^2C}{(1-\varepsilon)\delta} + \frac{nC^2}{\varepsilon(1-\varepsilon)\delta^2})$.

*Proof.* Let $\mathbb{P}_i$ be all augmenting paths found at the termination of phase $i$ and let $\mathbb{P}$ equal to the augmenting paths computed by the algorithm throughout all the phases. In a phase, the Hungarian search takes $\mathcal{O}(n^2)$ time as seen in 2.2.2. In the partial DFS step, every visited edge not on any augmenting path is deleted with only augmenting paths edges remaining and thus could be revisited. Therefore, for the partial DFS step in phase $i$ is $\mathcal{O}(n^2 + \sum_{P\in\mathbb{P}_i}|P|)$; here $|P|$ is length of augmenting path $P$. For all phases $\mathcal{O}(\frac{C}{\delta'})$ phases (according to Lemma 3.4), the total time $\mathcal{O}(\frac{C}{\delta'}n^2 + \sum_{P\in\mathbb{P}}|P|)$. Therefore due Lemma 3.5, the total time for all the scales is $\mathcal{O}(\frac{n^2C}{(1-\varepsilon)\delta} + \frac{nC^2}{\varepsilon(1-\varepsilon)\delta^2})$. □

## 3.2 Warm Start Algorithm

In this section, we present our algorithm for determining a $\delta$ additive approximation of the Optimal Transport problem as described in section 1.1. This algorithm adds a novel scaling algorithm which is adapted from Gabow and Tarjan's scaling paradigm on top of the $LMR$'s algorithm for scaled supplies, demands and costs as described in section 3.1.3. Given a transport problem $P$, the idea of this algorithm is as follows.

Using these dual weights as a **warm start**, we run the $LMR$ algorithm on the scaled transportation problem which is scaled by $\delta \leftarrow \frac{\delta}{2}$ and get another 1-feasible transport plan $\sigma_2$ and dual weights $y_2(\cdot)$. This process continues to iteratively scale the original capacities and costs by $\delta$, which is halved at each iteration. The iterations continue until a 1-feasible transport plan $\sigma$ and a corresponding set of dual weights $y(\cdot)$ are found for the capacities and costs scaled by the desired $\delta$. We can then use the procedure described in section 3.1.4 to get an approximate transport plan $\sigma$ that will have a desired additive error of $\delta$ (this is verified by Theorem 3.3).

Each iteration of the warm start algorithm with newly scaled capacities and costs is known as a scale. (Similar to a scale in Gabow and Tarjan's algorithm). Therefore this algorithm executes $\log\left(\frac{C}{\delta}\right)$ scales where $\delta$ is the desired additive error. Due to the use of warm starts, we observe that the cost of the transport plan found at the end of each scale is linear with respect to the total scaled supply at that scale which was not the case for $LMR$ algorithm as seen in Lemma 3.5. This property therefore allows the final execution time for the algorithm to be $\mathcal{O}\left(n^2 \cdot (\min\left(\sqrt{\frac{nC}{\delta}}, \frac{C}{\delta}\right)\right)$.

### 3.2.1 Definitions and Notations

The same definitions and notations used in section 3.1.1 are used here as well. Some additional definitions are the following

1. $\delta_i$ refers to the current value of delta at scale $i$. This parameter is fed into the graph theoretic algorithm described in 3.1.3. As we will see in the next section 3.2.3 $\delta_i = \frac{1}{2^{i-1}}$
2. $\mathcal{U}$ is the scaled total supply in the final scale of the algorithm. $\mathcal{U} = \Theta\left(\frac{2nC}{\epsilon\delta}\right)$ similar to how scaled supplies are defined in section 3.1.2. Similarly $\mathcal{U}_i$ is the total scaled supply at scale $i$ and is equivalent to $\mathcal{U}_i = \Theta\left(\frac{2nC}{\epsilon\delta_i}\right)$

### 3.2.2 The Algorithm

In this algorithm since the edge cost changes in every scale it is important to note that the $1 - feasibility$ of a transport plan discussed in this algorithm and the scaling algorithm would be defined with respect to

1. A set of dual weights
2. Edge cost at the current scale

**Initialization:** Initialize an empty transport plan $\sigma$ and set of dual weights $y(\cdot)$ set to 0 Set $\delta' = C$

Continue to do the following steps until $\delta' \leq \delta$:

1. **Scale Capacities and Costs**: Using the same scaling paradigm as $LMR's$ algorithm mentioned section 3.1.2 using the original transportation problem and $\delta'$. This outputs a scaled transportation problem

2. **Scaling Algorithm** Take a scaled transportation problem and set of initial dual weights and the scaling algorithm outputs a transport plan and set of dual weights that are 1-feasible with respect to the scaled costs of the scaled transportation problem

3. **Adjustment Step** Reduces dual weights of certain vertices to ensure that the slack on all edges carrying flow is at most 1 with respect to the scaled transportation problem.

4. **Update $\delta'$:** Set $\delta' \leftarrow \frac{\delta'}{2}$

### 3.2.3 Scaling Algorithm

Let $\sigma_i$ be a 1-feasible transport plan with respect to a set of dual weights $y_i(\cdot)$ jointly generated at the end of scale $i$ for integer scaled costs defined with respect to $\delta_i$ as described in section 3.1.2. Note that in scale 1 the empty transport plan with initial dual weights of 0 are 1-feasible for the scaled costs of scale 1. Then the steps of the algorithm as follows

Step 1: **Scaling Dual Weights and Costs**

In scale $i+1$ we set $\delta_{i+1} \leftarrow \frac{\delta_i}{2}$. This causes the scaling of costs and capacities as described in sections 3.1.1 and 3.1.2 respectively. Therefore if $c_i(a,b)$ is the scaled cost of an edge in the $i^{\text{th}}$ scale, then after cost scaling the scaled costs of all edges in scale $i+1$ with respect to the scaled cost of edges in scale $i$ is as follows:

$$2c_i(a,b) - 1 \leq c_{i+1}(a,b) \leq 2c_i(a,b)$$

Similarly the capacities of the $i+1^{\text{th}}$ scale with respect to the $i^{\text{th}}$ scale is as follows:

$$2\bar{s}_{b_i} \leq \bar{s}_{b_{i+1}} \leq 2\bar{s}_{b_i} + 1,$$

$$2\bar{d}_{a_i} - 1 \leq \bar{d}_{a_{i+1}} \leq 2\bar{d}_{a_i}.$$

Therefore in order to properly utilize $y_i(\cdot)$ as a warm start for the scaled transport problem in $i+1^{\text{th}}$ scale we need to create an initial set of 1-feasible dual weights, $y_{i+1}(\cdot)$ , with respect to $\sigma_i$ for the scaled costs of scale $i+1$ from $y_i(\cdot)$. For all vertices $v \in A \cup B$

$$y_{i+1}(v) \leftarrow 2y_i(v) - 1. \tag{3.13}$$

Step 2: **Conduct LMR's algorithm for scaled capacities on reduced edge costs**

In order to ensure that the transport plan generated throughout the execution scale $i+1$ is linear to the total scaled supplies the algorithm uses $y_{i+1}(\cdot)$ to calculate reduced costs. The *reduced cost* of any edge $(a, b)$ is defined as:

$$c'_{i+1}(a, b) = c_{i+1}(a, b) - y_{i+1}(a) - y_{i+1}(b).$$

Taking these scaled capacities, and reduced costs we conduct LMR's algorithm for scaled capacities as described in section 3.1.3 to compute a 1-feasible transport plan $\sigma_{i+1}$ and a set of dual weights $y'_{i+1}(\cdot)$ with respect to the reduced costs. Then for every vertex $v \in A \cup B$ the warm start algorithm does the following:

$$y_{i+1}(v) \leftarrow y_{i+1}(v) + y'_{i+1}(v)$$

in order to ensure that the transport plan $\sigma_i$ and the resultant $y_{i+1}(\cdot)$ are 1-feasible with respect to the original scaled costs $c_{i+1}(\cdot)$.

### 3.2.4 Adjustment Step:

Construct graph $G' = (A \cup B, E')$ where $E' = \{(a, b)|\ \sigma(a, b) \geq 0\}$ Then we the apply the reduction to vertices in $G'$ for vertices that meet both of the conditions:

1. Vertices that have a degree of 1 in $G'$.

2. The solitary incident edge to the vertex has a slack greater than 1.

The adjustment is as follows. Let vertex $v$ follow the above conditions and let the solitary edge $(u, v)$ have a slack, $s_i(u, v)$, greater than 1 then,

$$y_i(v) \leftarrow y_i(v) - s_i(u, v). \tag{3.14}$$

This reduction ensures that every backward edge has a slack of at most equal to 1. This ensures that the reduced edge costs produced in scale $i+1$ is 1-feasible with respect to scaled dual weights and scaled costs defined of scale $i + 1$.

### 3.2.5 Bipartite Matching via Warm Starts

Since Optimal Transport is a generalization of Bipartite Matching we can find fast additive approximations for Minimum Cost Maximum Cardinality Bipartite Matching by converting the above Optimal Transport Algorithm into a Matching Algorithm. Instead of repeatedly scaling capacities as described in section 3.2.3 each supply vertex can be assigned a supply of $\frac{1}{n}$ and each demand vertex can be assigned a demand of $\frac{1}{n}$. Then in every scale the original costs of transport problem $P$ would be repeatedly scaled based on the value of $\delta$ at that scale.

Therefore, considering a matching $M$, the corresponding transport plan has a cost given by:

$$\frac{1}{n}\sum_{(a,b)\in M} c(a,b) = \frac{1}{n}\cdot c(M)$$

To simplify the analysis, we can scale all demands and supplies from $\frac{1}{n}$ to 1 in each scale and then run $LMR$'s algorithm for scaled capacities and costs. So for any $\delta > 0$, scaling the demands and supplies from $\frac{1}{n}$ to 1 means that an $\delta$-approximate transport plan corresponds to a perfect matching $M$ with:

$$c(M) \leq c(M^*) + \delta n$$

This matching is now $\delta$-approximate matching. We show that this $\delta$-approximate matching can be found in $\mathcal{O}\left(n^2 \cdot min\left(\frac{C}{\delta}, \sqrt{n}\right)\right)$time. Note that $\mathcal{U}, \mathcal{D}$ are the final $\delta$ scaled supply and demands as defined in section. 3.1.2.

### 3.2.6 Correctness Proofs

**Lemma 3.7.** *Let $\overrightarrow{G}_\sigma$ be the residual graph at the end of scale $i$. Then if there exists a $(a,b) \in \overrightarrow{G}_\sigma(A,B)$ that has $s(a,b) > 1$ then the degree of $a$ or $b$ is equal to 1 in $\overrightarrow{G}_\sigma$.*

*Proof.* Assume for the sake of a contradiction that there exists an edge $(a,b) \in \overrightarrow{G}_\sigma(A,B)$ with $s(a',b') > 1$ where neither the degree of $a'$ nor the degree of $b'$ is one. Since the degree of $a'$ is greater than one this implies that $a'$ receives some mass from $b'$ and at least from one other arbitrary $\mathbf{b} \in B$ at the end of scaling algorithm. Similarly, since the degree of $b'$ is greater than one this implies that $b'$ sends some mass to $a'$ and to at least one other arbitrary $\mathbf{a} \in A$ at the end of scale $i$. Therefore there exists a forward edge and a backward edge between $a'$ and $b'$. Since $s(a',b') > 1$ this implies that $y(a) + y(b) > c(a,b) + 1$ which in turn would mean that the forward edge is infeasible. But the dual weights produced by the

$LMR$ algorithm (3.1.3) are 1-feasible therefore this is a contradiction. □

**Corollary 3.8.** *The reduction of dual weights described in the scaling paradigm still ensures 1-feasible dual weights for scale $i$.*

*Proof.* Let $(a, b) \in G'$ have a $s(a, b) > 1$. Then according to Lemma 3.7 either $a$ or $b$ have a degree of 1. Without any loss in generality assume that $a$ has a degree of 1. This would imply that there is only a backward edge between $(a, b)$ because if there were a forward edge then $y(a) + y(b) > c(a, b) + 1$ which is infeasible. Reducing $y(a)$ by $(y(a) + y(b) - c(a, b))$ ensures that $y(a) + y(b) = c(a, b)$ which is still 1-feasible according to feasibility condition (3.2). Since the degree of $a$ is one, reducing its dual weight does not make any other edge infeasible. Therefore, the resulting dual weights after the reduction procedure are still 1-feasible for scale $i$. □

**Theorem 3.9.** *The starting dual weights produced for scale $i+1$ is 1-feasible to scale $i+1$*

*Proof.* According to Corollary 3.8 the dual weights at the end of scale $i$ are 1-feasible to scale $i$. Due to the reduction process, all edges have a maximum slack of 1 in scale $i$. Since the costs of edges from scale $i$ to scale $i + 1$ are as follows

$$2c_i(a, b) - 1 \leq c_{i+1}(a, b) \leq 2c_i(a, b)$$

scaling dual-weights according to equation (3.13) ensures that $\sigma_i$ is 1-feasible with respect to costs $c_{i+1}(\cdot)$ and initial set of dual weights $y_{i+1}(\cdot)$. □

**Lemma 3.10.** *The transport plan $\sigma_i$ and the resultant dual weights $y_{i+1}(\cdot)$ produced at the end of scale $i + 1$ are 1-feasible with respect to the original scaled costs $c_{i+1}$ for scale $i$.*

*Proof.* At the beginning of scale $i+1$ the transport plan $\sigma_i$ is 1-feasible with respect to the the initial dual weights $y_{i+1}$ as seen in theorem 3.9. The $LMR$ algorithm produces a 1-feasible transport plan and set of dual weights $y'_{i+1}(\cdot)$. Dual weights $y'_{i+1}(\cdot)$ represent the growth in the original dual weights at the starting of the scale $i+1$, and therefore in step 2 of the scaling algorithm setting $y_{i+1}(v) \leftarrow y_{i+1}(v) + y'_{i+1}(v)$ ensures that the dual weights produced at the end of scale $i+1$ are 1-feasible to the transport plan $\sigma_i$ for scaled costs of scale $i+1$. □

### 3.2.7 Efficiency Proofs

The dual weight of demand vertices becomes smaller and smaller with every scale due to dual weight scaling described in equation (3.13). Therefore the reduced cost edges can grow larger that the original costs for some edges. The following proof ensures that the largest reduced scaled cost is not too large.

**Lemma 3.11.** *Given that the largest scaled cost of scale $i$ is $\frac{C}{\delta_i}$ then the largest reduced cost in scale $i$ is atmost $\frac{3C}{\delta_i}$ .*

*Proof.* Let's prove this claim by the principle of mathematical induction.

**Base Case:** In scale 1 the dual weights are initialized to zero. Therefore the diameter of the reduced costs is equal to the diameter of the original scaled costs at scale 1.

**Inductive Step:** Assume for scale $i \geq 1$ that if the largest scaled cost is $\frac{C}{\delta_i}$ then the largest reduced cost for scale $i$ is atmost $\frac{3C}{\delta_i}$. Let $\sigma$ be the transport plan found at the end of scale $i$. If edges $(a, b)$ had $\sigma(a, b) > 0$ at the end of scale $i$ then these edges would then lay on some augmenting path and have a reduced edge cost equal to 0. Therefore due to the cost scaling mechanism described in section 3.1.1 the reduced edge cost of these same edges would be some constant in scale $i+1$ therefore the claim would remain true.

Next we show that this claim is also true for all edges where $\sigma(a, b) = 0$. Let $a' \in A$ and $b' \in B$ be vertices such that $\sigma(a', b) \geq 0$ and $\sigma(a', b) \geq 0$ at end of scale $i$. Let $s(a, b)$ denote the reduced edge cost for edge $(a, b)$ in scale $i + 1$. Then

$$y(a') + y(b) + s(a', b) + y(a) + y(b') + s(a, b') = c(a', b) + c(a, b') \tag{3.15}$$

Due to the feasibility conditions

$$y(a') + y(b') \leq c(a', b') \tag{3.16}$$

Then from equations (3.16) and (3.15) we get the following:

$$y(b) + s(a', b) + y(a) + s(a, b') + c(a', b') \geq c(a', b) + c(a, b') \tag{3.17}$$

Rearranging terms

$$y(b) + y(a) + (s(a', b) + s(a, b') + c(a', b') - c(a', b) - c(a, b')) \geq 0 \tag{3.18}$$

Adding $c(a, b)$ on both sides we get

$$y(b) + y(a) + (s(a', b) + s(a, b') + c(a', b') - c(a', b) - c(a, b') + c(a, b)) \geq c(a, b) \tag{3.19}$$

By the defintion of the reduced edge costs we get

$$s(a, b) \leq s(a', b) + s(a, b') + c(a', b') - c(a', b) - c(a, b') + c(a, b) \tag{3.20}$$

Now $s(a', b)$ and $s(a, b')$ are upperbounded by some constant $c$ since they contained positive

flow in scale $i$ and $c(a', b') + c(a, b) \leq \frac{2C}{\delta_{i+1}}$ therefore our claim holds:

$$s(a, b) \leq 2c + \frac{2C}{\delta_{i+1}} \leq \frac{3C}{\delta_{i+1}}$$

Due to scaling mechanism and the fact that the input transportation problem may be an unbalanced transportation problem there might be demand $a \in A$ vertex that receives no mass in the scale $i$, but would receive mass in scale $i + 1$. If vertex $a$ received mass in scales prior to scale $i$ then by the inductive hypothesis and the above argument all edges incident to $a$ would not violate the claim. Otherwise if $a$ received no mass in scale prior to scale $i + 1$ then the dual weight of $a$ would be 0. Since the dual weight of every supply vertex is upper bounded by the diameter of the reduced cost space, the reduced edge costs of all the edges incident to $a$ would again not violate the claim. □

Therefore from the above lemma we can get our first bound for the number of phases in some arbitrary scale $i$.

**Corollary 3.12.** *The number of phases in scale $i$ is $\mathcal{O}\left(\frac{C}{\delta_i}\right)$.*

*Proof.* At the start of any phase there are no augmenting paths. During any phase let $b \in B_F$ be any free supply vertex. This implies that $b$ is a free supply vertex in every previous phase. The dual weight of a free supply vertex increases by atleast 1 at the end of every phase due to the Hungarian Search. Assume for the sake of a contradiction the algorithm runs for $\frac{3C}{\delta_i} + 2$ phases therefore allowing $y(b) \leq \frac{3C}{\delta_i} + 2$. Let $a \in A_F$ be a free demand vertex at the end of the $\frac{3C}{\delta_i} + 1$ phase. Therefore $a$ is saturated in the last phase and hence has a dual weight of 0 during the last phase. Then the forward edge $(a, b)$ would have $y(a) + y(b) \geq \frac{3C}{\delta_i} + 2$ and would hence violate 1-feasibility leading to a contradiction. □

The biggest advantage of using dual weights as warm starts is that the transport plan that

is produced using reduced costs at the end of every scale is small, which was not the case for $LMR$ algorithm

**Lemma 3.13.** *Let $\sigma_i$ be the transport plan produced at the end of scale $i$. Then, $w(\sigma_i) = \mathcal{O}(\mathcal{U}_i)$.*

*Proof.* To prove this claim, the principle of mathematical induction is applied.

**Base Case:** In the first scale, the dual weight of each vertex is initialized to zero, and the largest scaled edge cost is a constant, c (depending on the value of $\varepsilon$). Therefore, for scale one:

$$w(\sigma_1) = \sum_{(a,b)\in E} c(a,b)\cdot\sigma(a,b) \leq c \sum_{(a,b)\in E} \sigma(a,b) = c\cdot\mathcal{U}_1 = \mathcal{O}(\mathcal{U}_1)$$

**Inductive Step:** Assume that for scale $i$ (where $i \geq 1$), if $\sigma_i$ is the transport plan produced at the end of scale $i$, then $w(\sigma_i) = \mathcal{O}(\mathcal{U}_i)$. The total scaled supply that needs to be transported in scale $i+1$ is equal $\mathcal{U}_{i+1}$. In terms of $\mathcal{U}_i$, total scaled supply in scale $i+1$, $(\mathcal{U}_{i+1})$ can be defined as follows:

$$2\mathcal{U}_i \leq \mathcal{U}_{i+1} \leq 2\mathcal{U}_i + n$$

This is due to the capacity scaling paradigm as described in section 3.1.2. From this paradigm we get the following inequalities for the supply and demand capacities of individual vertices $b \in B$ and $a \in A$ in the $(i+1)$-th scale relative to the $i$-th scale:

$$\begin{aligned} 2\bar{s}_{b_i} \leq \bar{s}_{b_{i+1}} \leq 2\bar{s}_{b_i} + 1, \\ 2\bar{d}_{a_i} - 1 \leq \bar{d}_{a_{i+1}} \leq 2\bar{d}_{a_i}. \end{aligned} \tag{3.21}$$

Let $\mathcal{P}_i$ be the set of augmenting paths found during the execution of scale $i$. Then due to adjustment described in equation (3.14) in the the warm start algorithm description ( section 3.2.3) and the reduced costs at the beginning of scale $i+1$, there is a small constant distance

at the beginning of scale $i+1$ between the supply vertices and the demand vertices that were previously saturated at the end of scale $i$. The distance is constant and not zero due to the flooring mechanism described in cost scaling paradigm in section 3.1.1 and the additional $-1$ term in the dual weight scaling paradigm in equation (3.13) in section 3.2.3). Therefore after a constant number of phases ($s$) $2\mathcal{U}_i - n$ units of mass would have been transported. $2\mathcal{U}_i - n$ comes from the difference between the flooring and ceiling function as described in (3.21). Therefore a residual of $2n$ units of mass can be transported along augmenting paths with reduced costs of atmost $\frac{3C}{\delta_{i+1}}$ due Lemma 3.11. Therefore,

$$w(\sigma_{i+1}) = s(2\mathcal{U}_i - n) + \frac{6nC}{\delta_{i+1}} = \mathcal{O}(\mathcal{U}_{i+1})$$

□

The following lemma will help us find us find another bound for the number of phases in our algorithm.

**Lemma 3.14.** *At any point of the algorithm for a phase in arbitrary scale $i$, let $\mathbf{R}$ be the remaining supplies left to be transported and let $y_{max}$ be the current maximum dual weight. Then, $y_{max} \cdot \mathbf{R} = \mathcal{O}(\mathcal{U}_i)$.*

*Proof.* Let $\sigma_i$ be the final transport plan produced at the end of scale $i$ and let $\sigma_i^{'}$ be the transport plan at the end of the current phase. Similarly let $\mathcal{P}_i$ be the set of augmenting paths found during the entire execution of scale $i$ and $\mathcal{P}_i^{'}$ be the set of augmenting paths discovered at the end of the current phase. Using the above notations let's define the net cost of a transport plan ($\Phi(\sigma_i)$) as

$$\Phi(\sigma_i) = \sum_{P \in \mathcal{P}_i} r_P \Phi(P)$$

Recall that $\phi(P)$ is the net cost of an augmenting path $P$ (this notation is defined in Lemma 3.5) and $r_P$ is the units of supplies transported along augmenting path $P$. Then the difference between net costs of transport plans $\sigma_i$ and $\sigma_i^{'}$ is as follows:

$$\Phi(\sigma_i) - \Phi(\sigma_i^{'}) = \sum_{P \in \mathcal{P}_i} r_P \Phi(P) - \sum_{P \in \mathcal{P}_i^{'}} r_P \Phi(P) = w(\sigma_i) - w(\sigma_i^{'}) + \sum_{P \in (\mathcal{P}_i \backslash \mathcal{P}_i^{'})} r_P \cdot \left\lceil \frac{|P|}{2} \right\rceil \quad (3.22)$$

The third equality comes from the fact that

$$\begin{aligned} r_P \Phi(P) &= \Big( \sum_{(a',b') \in P^{\uparrow}} r_P \cdot \overline{c}(a',b') \Big) - \Big( \sum_{(a',b') \in P^{\downarrow}} r_P \cdot \overline{c}(a',b') \Big) + r_P \lceil |P|/2 \rceil & (3.23) \\ &= \overline{w}(\sigma') - \overline{w}(\sigma) + r_P \lceil |P|/2 \rceil & (3.24) \end{aligned}$$

where $\sigma^{'}$ is the transport plan after $r_P$ supplies have been transported across $P$, and $\sigma$ is before. Now according to Lemma 3.13 $\sigma_i = \mathcal{O}(\mathcal{U}_i)$. Therefore it is clear that $\sigma_i^{'} = \mathcal{O}(\mathcal{U}_i)$. From the equations we can get the following,

$$\sum_{P \in (\mathcal{P}_i \backslash \mathcal{P}_i^{'})} r_P \cdot \left\lceil \frac{|P|}{2} \right\rceil \leq \sum_{P \in (\mathcal{P}_i \backslash \mathcal{P}_i^{'})} r_P \cdot \Phi(P)$$

Note that $\sum_{P \in (\mathcal{P}_i \backslash \mathcal{P}_i^{'})} r_P \cdot \Phi(P) = \mathcal{O}(\mathcal{U}_i)$ Also, for $P \in (\mathcal{P}_i \backslash \mathcal{P}_i^{'})$, $\Phi(P) \geq y_{\max}$ where $y_{\max}$ is the current maximum dual weight at the end of the current phase. Similarly $\sum_{P \in (\mathcal{P}_i \backslash \mathcal{P}_i^{'})} r_P = \mathbf{R}$. Therefore, $y_{\max} \cdot \mathbf{R} = \mathcal{O}(\mathcal{U}_i)$. □

**Corollary 3.15.** *For a given scale $i$ in the the matching case, if $y_{max}$ is the maximum dual weight at the end of some arbitrary phase and $|B_F|$ is the number of free vertices remaining at the end of that phase. Then $y_{max}|B_F| = \mathcal{O}(n)$.*

*Proof.* The proof of this lemma is the same as the proof theorem 2.2 in [14] and for the sake of brevity as been discluded. The actual mechanics of the proof do not help in this thesis

only the result helps. □

**Lemma 3.16.** *The number of phases in scale $i$ is $\mathcal{O}(\sqrt{\mathcal{U}_i})$.*

*Proof.* Each augmenting path from a unsaturated supply vertex $b$ to a unsaturated demand vertex $a$ found in scale $i$ transports $r_p$ units of flow, where $r_p$ is defined as the minimum of the residual supply $b$, the residual demand at $a$, and the residual capacity of the bottleneck edge:

$$r_P = \min \left\{ s_b - \sum_{a' \in A} \sigma(a', b),\ d_a - \sum_{b' \in B} \sigma(a, b'),\ \mathrm{b}(P) \right\}.$$

This definition is provided in the augment procedure in section 3.1.3. Due to the properties of the Hungarian Search, $r_P \geq 1$. Additionally, due to the Hungarian Search step unless all of the supply has been transported at least one augmenting path is found at the end of every phase.

As a result of Lemma 3.14 after $\mathcal{O}(\sqrt{\mathcal{U}_i})$ phases, only $\mathcal{O}(\sqrt{\mathcal{U}_i})$ units of mass remains to be transported. Therefore, after $\mathcal{O}(\sqrt{\mathcal{U}_i})$ phases, all of the mass will have been transported. □

One of the advantages of using warm starts at the beginning of every scale is that the Hungarian Search algorithm does not need to spend a lot of iterations to find admissible augmenting paths in subsequent scales since previous scales have already done the work for you. Therefore the length of the augmenting paths found throughout this algorithm is much smaller than the length of the augmenting paths during the execution of the $LMR$ algorithm. This property therefore allows for us to get rid of the 2nd term that appears in the excution time of the $LMR$ algorithm.

**Lemma 3.17.** *Let $\mathcal{P}$ be the set of augmenting paths during the execution of the algorithm, then $\sum_{P \in \mathcal{P}} |P| = \mathcal{O}(\mathcal{U} \log(\mathcal{U}))$.*

*Proof.* From the proof of Lemma 3.5 we got the equation (3.12)

$$\sum_{P\in\mathcal{P}_i}(r_P\Phi(P)) = w(\sigma_i) + \sum_{P\in\mathcal{P}_i} r_P\lceil|P|/2\rceil \tag{3.25}$$

where $\sigma_i$ is the transport plan produced at the end of an arbitrary scale $i$. Since $w(\sigma_i) \geq 0$ and $r_P \geq 1$ therefore,

$$\sum_{P\in\mathcal{P}_i}\lceil|P|/2\rceil \leq \sum_{P\in\mathcal{P}_i}\Phi(P))$$

Therefore the length of all augmenting paths found throughout the warm start algorithm's execution is bounded by the total net cost of all of the augmenting paths found. The goal of this algorithm is to eventually transport $\mathcal{U}$ amounts of supply to the demand vertices, which is done in the last scale. Therefore Lemma 3.14 can be rewritten such that at any point of the algorithm for a phase, if $\mathbf{R}$ is the remaining supplies left to be transported and $y_{\max}$ is the current maximum dual weight at the end of some arbitrary phase in some arbitrary scale, $y_{\max}\cdot\mathbf{R} = \mathcal{O}(\mathcal{U})$. This extension follows because in any scale $\mathcal{U}_i \leq \mathcal{U}$ From the proof of Lemma 3.5 we know that for every augmenting path from some supply vertex $b$, the net cost of the path is equal to $y(b)$. Let $y_{\max}$ be the current maximum dual weight when augmenting $P$ at the end of some phase. Similarly let $R_i$ be the remaining supply when augmenting path $P$ has been augmented. Then $y_{\max}\cdot R_i = \mathcal{O}(\mathcal{U})$. Summing over all of the paths we get the following:

$$\sum_{P\in\mathcal{P}}(r_P\Phi(P)) = \mathcal{O}(\mathcal{U})\cdot\sum_{P\in\mathcal{P}}\frac{1}{R_i} = \mathcal{O}(\mathcal{U}\log(\mathcal{U}))$$

This above comes from the fact that $\mathcal{U}_i \leq \mathcal{U}$ therefore using Lemma 3.13 we can conclude that the transport plan at the end of the every has a cost of $\mathcal{O}(\mathcal{U})$ □

**Corollary 3.18.** *Let $\mathcal{P}$ be the set of augmenting paths found during the execution of the matching algorithm, then $\sum_{P\in\mathcal{P}}|P| = \mathcal{O}(n\log(n))$.*

*Proof.* Following the same logic of Lemma 3.17 and using corollary 3.15 and the fact that every matching found at the end of every scale has a cost of $\mathcal{O}(n)$ we can conclude that the length of all augmenting paths found for $\delta$-approximate matching is $\mathcal{O}(n \log(n))$ □

**Theorem 3.19.** *The running time of this warm starts algorithm is* $\mathcal{O}\left(n^2 \cdot (\min\left(\sqrt{\frac{nC}{\delta}}, \frac{C}{\delta}\right)\right)$.

*Proof.* To bound the execution time of a single scale $i$ of the algorithm, the bounds established in Corollary 3.12 and Lemma 3.16 are used. These results show that the total number of phases in scale $i$ is given by

$$\min\left(3\left\lfloor\frac{C}{1-\epsilon}\cdot 2^{i-1}\right\rfloor, \sqrt{\mathcal{U}_i}\right),$$

with the largest reduced cost in scale $i$ being $\left\lfloor\frac{C}{1-\epsilon}\cdot 2^{i-1}\right\rfloor$. The term $2^{i-1}$ comes from the fact that delta is halved for every scale. Within each phase, the Hungarian search step takes $\mathcal{O}(n^2)$. Similarly partial DFS depends on the length of all the augmenting paths found in that phase. Therefore, across all phases of scale $i$ the total time taken is $\mathcal{O}\left(3\left\lfloor\frac{C}{1-\epsilon}\cdot 2^{i-1}\right\rfloor n^2 + \sum_{P\in\mathcal{P}_i}|P|\right)$. Using Lemma 3.17 we know that the length of augmenting paths produced throughout the algorithm is $\mathcal{O}(\mathcal{U}\log(\mathcal{U}))$. The total running time of scale $i$ is equal to $\mathcal{O}\left(\min\left(3\left\lfloor\frac{C}{1-\epsilon}\cdot 2^{i-1}\right\rfloor, \sqrt{\mathcal{U}_i}\right) n^2 + \sum_{P\in\mathcal{P}_i}|P|\right)$

Summing across all scales we get the following:

$$\begin{aligned}
&\sum_{i=1}^{\log(\frac{C}{\delta})} \mathcal{O}\left(\min\left(3\left\lfloor\frac{C}{1-\epsilon}\cdot 2^{i-1}\right\rfloor, \sqrt{\mathcal{U}_i}\right) n^2 + \sum_{P\in\mathcal{P}_i}|P|\right) \\
&= \mathcal{O}\left(\min\left(\frac{C(1-\delta)}{\delta(1-\epsilon)}, \sqrt{\frac{nC(1-\delta)}{\delta(1-\epsilon)}}\right) n^2 + \frac{nC}{\epsilon\delta}\log\left(\frac{nC}{\epsilon\delta}\right)\right)
\end{aligned} \tag{3.26}$$

This is equivalent to $\mathcal{O}\left(n^2 \cdot (\min\left(\sqrt{\frac{nC}{\delta}}, \frac{C}{\delta}\right)\right)$ □

**Corollary 3.20.** *The running time of this warm starts algorithm for matching is* $\mathcal{O}\left(n^2 \cdot min\left(\frac{C}{\delta}, \sqrt{n}\right)\right)$.

*Proof.* To bound the execution time of a single scale $i$ of the algorithm we bound the number of phases by the diameter of any scale $\frac{C}{\delta}$. Similar to Gabow and Tarjan we can bound the number of phases by $\mathcal{O}(\sqrt{n})$. In every phase the dual weight of $B$ increases by at least one due to the Hungarian Search (section 2.2). Since the matching size decreases by 1 similar to the Hungarian Algorithm the number of free $B$ vertices also reduces by 1 after every phase. Therefore after $\sqrt{n}$ phases, there are $\sqrt{n}$ B vertices yet to be matched due to Lemma 3.15. Therefore in $2\sqrt{n}$ phases all of the B vertices are matched. Hence, the number of phases is also bounded by $\mathcal{O}(\sqrt{n})$.

These results show that the total number of phases in scale $i$ is given by

$$\min\left(\frac{C}{\delta}, \sqrt{n}\right),$$

Following a similar logic to Theorem 3.19 we can get that across all phases of scale $i$ the total time taken is $\mathcal{O}\left(\min\left(\frac{C}{\delta}, \sqrt{n}\right) n^2 + \sum_{P \in \mathcal{P}_i} |P|\right)$. Using Lemma 3.17 and 3.15 we bound the total length of the augmenting paths found in the execution of the entire algorithm to $\mathcal{O}(n \log(n))$. The total running time of scale $i$ is equal to $\mathcal{O}\left(\min\left(\frac{C}{\delta}, \sqrt{n}\right) n^2 + \sum_{P \in \mathcal{P}_i} |P|\right)$ Summing across all scales we get the following:

$$\sum_{i=1}^{\log(\frac{C}{\delta})} \mathcal{O}\left(\min\left(\frac{C}{\delta}, \sqrt{n}\right) n^2 + \sum_{P \in \mathcal{P}_i} |P|\right) = \mathcal{O}\left(n^2 \cdot min\left(\frac{C}{\delta}, \sqrt{n}\right)\right) \tag{3.27}$$

□

## 3.3 Experimental Results

### 3.3.1 Dataset and Experiments:

The process begins by randomly generating two sets of points, $A$ and $B$, each consisting of $n$ points within the unit square $[0,1] \times [0,1]$. These points represent the supply and demand locations in the optimal transport problem. Each point is assigned an equal mass of $\frac{1}{n}$, ensuring the total mass of both sets equals one effectively making this a matching problem. The cost matrix $C$ is then computed based on the Euclidean distance between each pair of points in $A$ and $B$. The entries in $C$ are normalized so that the maximum distance is scaled to 1.

For the OT problem, these points are given mass from a uniform distribution $\mathcal{U}[0,1]$ and then normalized.

We averaged over 100 runs, where each run executes our algorithm on randomly generated points as described above. Then for $\delta \in [10^{-6}, 10^{-3}]$ we record the iteration count or the total number of phases of our algorithm. Since the underlying implementation of each scale is the $LMR$ algorithm, speedup can be identified via the number of iterations, i.e, fewer phases indicates faster algorithm. Results can be seen in Figure 3.1

### 3.3.2 Results

#### Explanation of Results

As $\delta$ grows smaller ($\delta \to 0$) the $LMR$ algorithm's dominant term $\mathcal{O}\left(n^2\left(\frac{C}{\delta}\right)\right)$ tends towards $\mathcal{O}(n^3)$ since the diameter $\left(\frac{C}{\delta}\right)$ grows larger and larger (Note in the matching problem the 2nd term is not relevant). But in warm starts algorithm the runtime is $\mathcal{O}\left(n^2 \cdot \min\left(\frac{C}{\delta}, \sqrt{n}\right)\right)$therefore

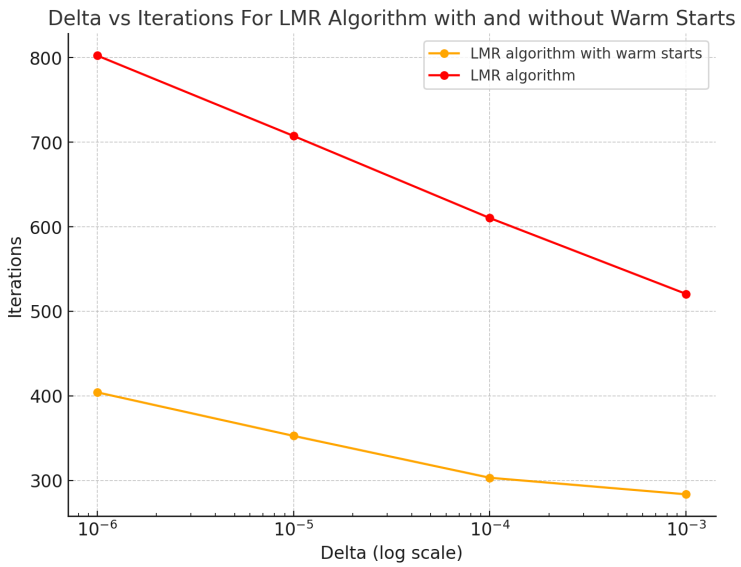


Figure 3.1: Delta vs Iterations for Matching: $LMR$ algorithm with warm starts and $LMR$ algorithm.

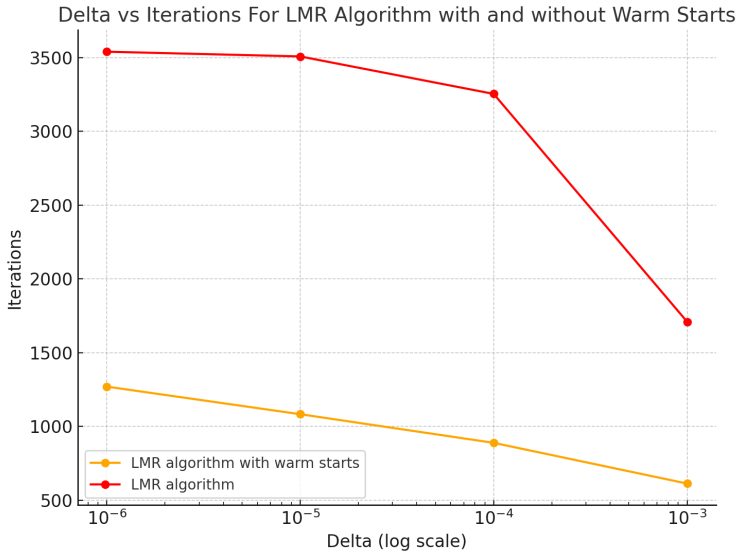


Figure 3.2: Delta vs Iterations for OT: $LMR$ algorithm with warm starts and $LMR$ algorithm.

for extremely small $\delta$ values in the worst case this term only becomes $\mathcal{O}(n^{2.5})$ which can be shown in this speedup as the number of iteration is significantly less for smaller $\delta$ values.

For optimal transport (OT), as $\delta \to 0$, the $LMR$ algorithm with a running time of $\mathcal{O}\left(n^2\left(\frac{C}{\delta}\right) + n\left(\frac{C}{\delta}\right)^2\right)$

(from [20]) sees the second term becoming more dominant. This is because the diameter, $\frac{C}{\delta}$, grows larger, and in the second term the diameter is squared, making its contribution significantly more pronounced with it's running time tending towards $\mathcal{O}(n^3)$. In contrast, the warm start algorithm, with a running time of $\mathcal{O}\left(n^2 \cdot \min\left(\sqrt{\frac{nC}{\delta}}, \frac{C}{\delta}\right)\right)$, does not suffer from this issue as it grows towards $\mathcal{O}(n^3)$ more slowly. As a result, its growth is much slower for higher $\delta$ values and therefore takes fewer iterations for small $\delta$ values.

# Chapter 4

# Learning Augmented Algorithms

Learning-augmented algorithms integrate machine learning predictions to enhance the efficiency and accuracy of classical optimization algorithms. By leveraging predictions to **warm-start** the process, these algorithms initialize closer to an optimal solution, potentially lowering runtime and resource usage.

In problems like Minimum Cost Bipartite Matching and Optimal Transport, learning-augmented algorithms aim to predict dual weights that can be used as warm starts to guide the initialization closer to optimality. Dual weights are the chosen entity to predict because they come from the dual formulation of the problem's linear programming (LP) relaxation, and they encode valuable information about the solution structure, often guiding algorithms directly toward optimal solutions or feasible regions that are close to optimal. Therefore during execution, we see a reduction in the number of steps or iterations required to reach a solution, leading to faster and more efficient performance.

## 4.1 Prior Algorithm for Minimum Cost Matching

This section presents the current best algorithm for solving the Minimum-Weight Bipartite Perfect Matching problem given a set of learned dual weights: the algorithm using a primal-dual scheme by (Chen et al, 2022 [8]). By leveraging learned predictions, this approach aims to improve upon classical algorithms, such as the Hungarian method, in terms of runtime.

### 4.1.1 Learning Good Dual Weights

According to Theorem 1 of Dinitz et al. [10],

> "There exists a single algorithm that, with access to $\mathcal{O}(C^2n^3)$ problem instance samples from an unknown distribution $D$, achieves an expected running time on future instances from $D$ of:
>
> $$\mathcal{O}(m\sqrt{n}\min\{\alpha, \sqrt{n}\}),$$
>
> where $\alpha = \min_y \mathbb{E}_{c\sim D}[\|y - y^*(c)\|_1]$".

This single algorithm combines the following three sub-algorithms to achieve faster matching:

1. **Learning Algorithm**: Utilizes an empirical risk minimization strategy on a training set of $\mathcal{O}(C^2n^3)$ samples to learn effective dual weights.

2. **Feasibility Algorithm**: Employs a greedy approximation algorithm running in $\mathcal{O}(m+n)$ to adjust the learned dual weights, ensuring they are feasible according to equations (2.1) and (2.2), as detailed in Section 2.2.2.

3. **Optimization Algorithm**: The optimization algorithm presented in section 4.1.2 is from Chen et al, 2022 [8] which is the current fastest optimization algorithm for minimum cost bipartite matching at the time of writing this thesis.

Details of the Learning and Feasibility Algorithms are not critical for this thesis and are omitted here.

### 4.1.2 Optimization Algorithm

The algorithm starts by identifying edges tight under the given dual predictions, $y(\cdot)$. An edge is tight if the sum of the given predicted dual weights is equal to the cost of the edge (exactly the same as (2.2)) These tight edges, forming a subgraph (residual graph), are used to compute an initial maximum cardinality matching. To represent the matching problem as a flow network, all edges are assigned unit capacities and directed from left to right. Additionally, source and sink nodes are added, and connected with appropriate zero-cost edges. Dual variables are initialized as follows:

$$\begin{aligned} z_i &= -y_i \qquad \forall i \in L \\ z_j &= y_j \qquad \forall i \in R \end{aligned} \tag{4.1}$$

Based on these initialized dual variables reduced edge costs are defined as follows:

$$c'(i,j) = c'(i,j) + z_i - z_j \qquad \forall (i,j) \in E \tag{4.2}$$

The algorithm iteratively adjusts the dual variables based on shortest path computations found via Dijkstra's algorithm in the residual graph to ensure the feasibility of the updated matching. Specifically, the shortest path distances from the source to other nodes are used to incrementally adjust the dual variables. This adjustment maintains reduced costs for all residual edges. The algorithm then augments the flow along the identified zero-reduced cost path similar to Ford Fulkerson's algorithm. This step increases the total flow by one unit, moving the solution closer to achieving a perfect matching. Finding zero reduced cost paths and updating dual weights is the same as a Hungarian Search as seen in section 2.2 The iteration proceeds until the total flow equals the number of vertices, guaranteeing a perfect matching. The running time of this algorithm is given $\mathcal{O}\left(m\sqrt{n} + (m + n\log n)\|y^* - \hat{y}'\|_0\right)$

where $\|y^* - \hat{y}'\|_0 = o(n)$ denotes the sparsity of the difference between the optimal dual and the predicted dual variables.

## 4.2 Our Optimization Algorithm

Our optimization algorithm uses the assumption that the $l_1$-norm error of the learned duals is $o(\sqrt{n})$ which is the same assumption taken and shown by Dinitz et al. [10] to be possible.

A simple modification to Chen's et al. [8] optimization algorithm comes from applying Gabow's Algorithm [13]. Instead of augmenting along a single augmenting path as done above, Gabow's Algorithm augments along all of the possible augmenting paths found at the end of one iteration with the help of the Hopcroft-Karp algorithm as shown in 2.1. The steps of the algorithm are detailed as follows: Initially, the algorithm starts with an empty matching $M$ and initializes the dual weights $y$ to the predicted values $\hat{y}$. These dual weights provide a starting estimate for the optimal solution and can significantly reduce the number of iterations required for convergence. The main computation is performed iteratively within a `while` loop, which continues until $M$ becomes a perfect matching on the graph $G$. A perfect matching is achieved when every vertex in one partition of the bipartite graph is matched to exactly one vertex in the other partition.

**Step 1: Hungarian Search for an Augmented Path.** In each iteration, the algorithm conducts a Hungarian search to identify an augmenting path. This step is exactly the same as described in section 2.2.2

**Step 2: Constructing the Reduced Graph $G'$.** Using the updated dual weights from the previous step, the algorithm constructs a reduced graph $G' = (V, E')$, where the edge set $E'$ consists of all edges $(u, v)$ in $G$ with a reduced cost $s(u, v)$ equal to zero. The reduced

cost is calculated as:

$$s(u, v) = c(u, v) - y(u) - y(v),$$

where $c(u, v)$ is the original cost of the edge, and $y(u)$ and $y(v)$ are the dual weights of the respective vertices. The reduced graph $G'$ captures the essential structure for finding a maximum cardinality matching that preserves dual feasibility.

**Step 3: Maximum Cardinality Matching on $G'$.** With the reduced graph $G'$ constructed, the algorithm employs the Hopcroft-Karp algorithm to compute a maximum cardinality matching on $G'$. The Hopcroft-Karp algorithm uses BFS and DFS as stated in section 2.1 to increase the matching size more than 1 in each iteration. The algorithm repeats these steps, progressively refining the matching and dual weights, until $M$ becomes a feasible matching with respect to the final dual weights and therefore a perfect matching. Once this condition is satisfied, the `while` loop terminates, and the algorithm returns the perfect matching $M$ along with the corresponding dual variables $y$.

The steps discussed above are formalized in Algorithm 1, which outlines the execution of Gabow's algorithm using predicted dual weights for efficient bipartite matching:

**Algorithm 1** Gabow's algorithm with predicted dual weights

$M \leftarrow \emptyset$
$y \leftarrow \hat{y}$
**while** M is not a perfect matching on G **do**:
    Step 1.1: Conduct a Hungarian Search to find an augmented path
    Step 1.2: Construct G' = (V,E') where E' = {(u,v) | s(u,v) = 0}.
    Step 1.3: Populate M by conducting a maximum cardinality matching on G' by the Hopcroft-Karp algorithm
**end while**
Return the perfect matching M and corresponding dual variables

### 4.2.1 Running Time Proofs

Let the minimum-cost perfect cardinality matching be $M^*$. Let $c(M^*)$ denote the cost of matching $M^*$ where cost of a matching is defined to be the sum of the costs of all the edges in $M^*$. Let there be $n$ vertices and $m$ edges in the bipartite graph. Note that the number of edges is $\mathcal{O}(n^2)$.

Let $\hat{y}$ be the set of given feasible predicted dual weights and y* be the optimal dual weights corresponding to M* for graph $G$. Then we can define $I$ (the gap between the current state of matching and final optimal matching) to be the following:

$$I = c(M^*) - \sum_{i \in V} \hat{y}(i).$$

**Lemma 4.1.**

$$I \leq ||y^* - \hat{y}||_1.$$

*Proof.* Since every edge in M* has 0 slack

$$c(M^*) = \sum_{i \in V} y^*(i).$$

Therefore,

$$I = \sum_{i \in V} y^*(i) - \hat{y}(i) \leq \sum_{i \in V} |y^*(i) - \hat{y}(i)| = ||y^* - \hat{y}||_1$$

Hence,

$$I \leq ||y^* - \hat{y}||_1.$$

□

At the beginning of any given iteration k let the current matching be M. Let there be $f_k$ free vertices with respect to M at the beginning of iteration k. At each iteration the Hungarian Search alters the dual variables by $\delta_k$.

Define $\Delta$:

$$\Delta = \sum_{i=1}^{k} \delta_i$$

**Lemma 4.2.** $f_k \cdot \Delta \leq I$

*Proof.* Define X to be the following:

$$X = \sum_{i \in V} \hat{y}(i)$$

At any iteration i dual weights are adjusted by $\delta_i$ causing X to increase by $f_i \cdot \delta_i$.

X increases by $f_i \cdot \delta_i$ because all vertices in M do not contribute to the increase in X. In the case u is matched ($u \in M$) to some vertex v there is an increase in dual weight $y(u)$ by $\delta_i$ (due to Hungarian Search). Still, in order to keep $y(v)$ feasible (so that M remains feasible) there is a decrease of $y(v)$ by $\delta_i$ (Also done by Hungarian Search) hence keeping X the same. But if u is a free vertex then it increases X by $\delta_i$.

Now every augment decreases $f_i$ by 2, so if $i < k$ then $f_i \geq f_k$. Therefore the total increase of X up to iteration k is at least as follows:

$$f_k \cdot \sum_{i=1}^{k} \delta_i \ = f_k \cdot \Delta$$

But, X increases at most by I hence,

$$f_k \cdot \Delta \leq I$$

□

**Theorem 4.3.** *The time complexity of this algorithm is* $\mathcal{O}(\sqrt[4]{||y^* - \hat{y}||_1} \cdot n^{1/2}m)$ *and the space complexity is* $\mathcal{O}(m)$ *where* $n$ *is the number of vertices and* $m$ *is the number of edges.*

*Proof.* According to Lemma 4.1 and Lemma 4.2 for any iteration $k$ we observe the following:

$$f_k \cdot \Delta \leq I \leq ||y^* - \hat{y}||_1$$

Letting $f_k \geq \sqrt{||y^* - \hat{y}||_1}$ we get the following:

$$\Delta \leq \frac{I}{f_k} \leq \sqrt{||y^* - \hat{y}||_1}$$

Every iteration i of Step 1 of the algorithm adjusts the dual weights by some positive $\delta_i$. Note that $\delta_i \geq 1$ since edge costs and dual weights are integral. Thus if $f_k \geq \sqrt{||y^* - \hat{y}||_1}$ then k is atmost $\sqrt{||y^* - \hat{y}||_1}$. The number of executions with $f_k \leq \sqrt{||y^* - \hat{y}||_1}$ is less than, $\frac{1}{2} \cdot \sqrt{||y^* - \hat{y}||_1}$ since each execution matches two or more free vertices. Therefore Step 1 is executed less than $2\sqrt{||y^* - \hat{y}||_1}$

Similarly, as in the above argument, we can conclude that step 1 of the algorithm is executed at most $\sqrt[4]{||y^* - \hat{y}||_1}$ times with $f_k \geq ||y^* - \hat{y}||_1^{0.375}$.

Now to compute the total time of steps 1, 2, and 3:

**Step 1**: Dijkstra's algorithm is performed in $\mathcal{O}(m)$ with help of special data structures such as Fibonacci heaps. Therefore the total time to execute step 1.1 is $\mathcal{O}(\sqrt{||y^* - \hat{y}||_1}m)$

**Step 2 and 3**: One execution of Hopcraft-Karp is $\mathcal{O}(\min(n^{0.5}, a)m)$. Here "a" is the number of augmenting paths found (the O(am) bound follows from inspecting the Hopcroft-Karp algorithm). With $f \geq ||y^* - \hat{y}||_1^{0.375}$ the time of all executions of steps 1.2 and 1.3 is $\mathcal{O}(\sqrt[4]{||y^* - \hat{y}||_1} \cdot n^{1/2}m)$. The executions with $f_k < ||y^* - \hat{y}||_1^{0.375}$ has less than $\frac{1}{2} \cdot ||y^* - \hat{y}||_1^{0.375}$

augmenting paths so $\mathcal{O}(\sqrt[4]{||y^* - \hat{y}||_1} \cdot n^{1/2}m)$ is appropriate. Note that a feasible dual being within $\mathcal{O}(\sqrt{n})$ of an optimal dual in $l_1$ distance is possible according [10] hence it is fair to assume $||y^* - \hat{y}||_1 \simeq n^{1/2}$. Therefore Algorithm time complexity is $\mathcal{O}(\sqrt[4]{||y^* - \hat{y}||_1} \cdot n^{1/2}m)$ □

The following corollary shows that with better and better predictions we could tend towards $\mathcal{O}(\sqrt{n}m) = \mathcal{O}(n^{2.5})$

**Corollary 4.4.** *The time complexity of this algorithm is* $\mathcal{O}(\sqrt[4]{I} \cdot n^{1/2}m)$ *and the space complexity is* $\mathcal{O}(m)$ *where* $n$ *is the number of vertices and* $m$ *is the number of edges.*

*Proof.* From Lemma 4.2 we have that for any iteration $k$ of the algorithm

$$f_k \cdot \Delta \leq I$$

Therefore the number of executions of step 1 satisfies two properties:

a. Step 1 is executed less than $2 \cdot \sqrt{I}$

b. Step 1 is executed at most $\sqrt[4]{I}$ when $f_k \geq I^{\frac{3}{4}}$

Property a. follows from Theorem 4.3: If $f_k \geq \sqrt{I}$ then $k$ is at most $\sqrt{I}$ (Since dual weights in every iteration are adjusted by at least 1). And if $f_k \leq \sqrt{I}$ then $k$ is less than $\frac{\sqrt{I}}{2}$ since each iteration matches two or more free vertices. Property b. Follows similar: If $f_k \geq I^{\frac{3}{4}}$ then $\Delta \leq \sqrt[4]{I}$ hence $k \leq \sqrt[4]{I}$ Therefore from the discussions in Theorem 4.3 and Property a the total time taken for step 1 is $\mathcal{O}(\sqrt{I} \cdot m)$ From the discussion in theorem and property b the total time taken by Step 2 and Step 3 $\mathcal{O}(\sqrt[4]{I} \cdot n^{1/2}m)$ Hence total time taken for the algorithm is $\mathcal{O}(\sqrt[4]{I} \cdot n^{1/2}m)$. □

# Chapter 5

# Conclusions

In this thesis we have devised two distinct algorithms, both utilizing dual weights as warm starts to compute a minimum-cost maximum cardinality matching. Notably, one of these algorithms is also capable of finding an optimal transport plan using dual weights as warm starts.

The first algorithm introduces a novel scaling approach that leverages dual weights obtained from simplified versions of the same transport or matching problem. Dual weights derived from multiple executions of the $LMR$ algorithm [20], for varying values of $\delta$, are utilized to achieve the desired $\delta$-precision. The theoretic and experimental results show that this algorithm can be used to provide a smooth tradeoff between approximate and exact algorithms due to it's reduced number of iterations for extremely small $\delta$ values.

The first algorithm can also be extended from the matching problem to the transport problem. The warm start algorithm shows improvement in running time for extremely small $\delta$ values such as $\delta < 10^{-3}$ as compared to the $LMR$ algorithm which was the previous best sequential additive approximation of optimal transport. For extremely small $\delta$ values the $LMR$ algorithm tends towards a running time similar to the Hungarian Algorithm which is not the case for the warm start algorithm. This improvement stems from the manageable size of each transport plan at each scale, made possible by the warm start initialization derived from the solution at the previous scale.

The second algorithm is a learning-augmented method that utilizes machine-learned dual weights derived from techniques stated in prior work, [10] to achieve optimal matching with greater efficiency than existing learning-augmented techniques for the same problem. The improvement stems from the observation that, instead of increasing the matching size by one per iteration by augmenting along a single augmenting path, it is possible to augment along multiple augmenting paths per iteration. This approach reduces the total number of iterations required. By leveraging the Hopcroft-Karp technique, this algorithm improves upon existing learning-augmented bounds for solving the minimum weight bipartite matching problem.

We conclude this thesis with the following open problems:

- While the learning-augmented algorithm theoretically provides a speedup, it faces practical limitations. As shown by Dinitz et al, [10], learning dual weights from an unknown distribution $D$ that require $\mathcal{O}(\sqrt{n})$ phases to reach optimality necessitates $\mathcal{O}(C^2n^3)$ samples. Furthermore, ensuring the feasibility of these start dual weights involves an additional $\mathcal{O}(n^2)$ projection algorithm. Therefore can we come up with a faster way to predict good dual weights?

- Can we extend the Warm Start approach from the $LMR$ algorithm to a parallel framework like PyCoOT [21]?

- Identify real-world datasets where the matching and optimal transport version of the first algorithm demonstrates significant performance improvements over prior works similar to the synthetic dataset shown here.

# Bibliography


[1] Pankaj K. Agarwal and R. Sharathkumar. Approximation algorithms for bipartite matching with metric and geometric costs. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 555–564, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450327107. doi: 10.1145/2591796.2591844. URL https://doi.org/10.1145/2591796.2591844.

[2] Keerti Anand, Rong Ge, and Debmalya Panigrahi. Customizing ml predictions for online algorithms, 2022. URL https://arxiv.org/abs/2205.08715.

[3] Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. *CoRR*, abs/2003.02144, 2020. URL https://arxiv.org/abs/2003.02144.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017. URL https://arxiv.org/abs/1701.07875.

[5] Etienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20083–20094. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/e834cb114d33f729dbc9c7fb0c6bb607-Paper.pdf.

[6] Jean-David Benamou, Guillaume Carlier, Marco Cuturi, Luca Nenna, and Gabriel Peyré. Iterative bregman projections for regularized transportation problems, 2014. URL https://arxiv.org/abs/1412.5154.

[7] Jérémie Bigot, Raúl Gouet, Thierry Klein, and Alfredo López. Geodesic pca in the wasserstein space, 2014. URL https://arxiv.org/abs/1307.7721.

[8] Justin Chen, Sandeep Silwal, Ali Vakilian, and Fred Zhang. Faster fundamental graph algorithms via learned predictions. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 3583–3602. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/chen22v.html.

[9] Marco Cuturi and Arnaud Doucet. Fast computation of wasserstein barycenters, 2014. URL https://arxiv.org/abs/1310.4375.

[10] Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Faster matchings via learned duals. *CoRR*, abs/2107.09770, 2021. URL https://arxiv.org/abs/2107.09770.

[11] Paul Dütting, Silvio Lattanzi, Renato Paes Leme, and Sergei Vassilvitskii. Secretaries with advice, 2020. URL https://arxiv.org/abs/2011.06726.

[12] Rémi Flamary, Marco Cuturi, Nicolas Courty, and Alain Rakotomamonjy. Wasserstein discriminant analysis. *Machine Learning*, 107(12):1923–1945, May 2018. ISSN 1573-0565. doi: 10.1007/s10994-018-5717-1. URL http://dx.doi.org/10.1007/s10994-018-5717-1.

[13] Harold N. Gabow. Scaling algorithms for network problems. *J. Comput. Syst. Sci.*, 31(2):148–168, 1985. doi: 10.1016/0022-0000(85)90039-X. URL https://doi.org/10.1016/0022-0000(85)90039-X.

[14] Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for network problems.

*SIAM Journal on Computing*, 18(5):1013–1036, 1989. doi: 10.1137/0218069. URL https://doi.org/10.1137/0218069.

[15] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi: 10.1137/0202019. URL https://doi.org/10.1137/0202019.

[16] Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online Algorithms for Weighted Paging with Predictions. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 69:1–69:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-138-2. doi: 10.4230/LIPIcs.ICALP.2020.69. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2020.69.

[17] Andrey Boris Khesin, Aleksandar Nikolov, and Dmitry Paramonov. Preconditioning for the geometric transportation problem. *CoRR*, abs/1902.08384, 2019. URL http://arxiv.org/abs/1902.08384.

[18] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. doi: https://doi.org/10.1002/nav.3800020109. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109.

[19] Nathaniel Lahn and Sharath Raghvendra. A weighted approach to the maximum cardinality bipartite matching problem with applications in geometric settings. *CoRR*, abs/1903.10445, 2019. URL http://arxiv.org/abs/1903.10445.

[20] Nathaniel Lahn, Deepika Mulchandani, and Sharath Raghvendra. A graph theoretic additive approximation of optimal transport. *CoRR*, abs/1905.11830, 2019. URL http://arxiv.org/abs/1905.11830.

[21] Nathaniel Lahn, Sharath Raghvendra, and Kaiyi Zhang. A combinatorial algorithm for approximating the optimal transport in the parallel and mpc settings. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 21675–21686. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/448444518637da106d978ae7409d9789-Paper-Conference.pdf.

[22] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, page 1859–1877, USA, 2020. Society for Industrial and Applied Mathematics.

[23] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice, 2020. URL https://arxiv.org/abs/1802.05399.

[24] Jeff Phillips and Pankaj Agarwal. On bipartite matching under the rms distance. 01 2006.

[25] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/73a427badebe0e32caa2e1fc7530b7f3-Paper.pdf.

[26] Sharath Raghvendra and Pankaj K. Agarwal. A near-linear time -approximation algorithm for geometric bipartite matching. *J. ACM*, 67(3), June 2020. ISSN 0004-5411. doi: 10.1145/3393694. URL https://doi.org/10.1145/3393694.

[27] Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice, 2019. URL https://arxiv.org/abs/1910.12172.

[28] Roman Sandler and Michael Lindenbaum. Nonnegative matrix factorization with earth mover's distance metric for image analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1590–1602, 2011. doi: 10.1109/TPAMI.2011.18.

[29] R. Sharathkumar and Pankaj K. Agarwal. Algorithms for the transportation problem in geometric settings. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, page 306–317, USA, 2012. Society for Industrial and Applied Mathematics.