A Novel Skip Orthogonal List for Dynamic Optimal Transport Problem

Xiaoyang Xu, Hu Ding

University of Science and Technology of China
xiaoyangxu@mail.ustc.edu.cnhuding@ustc.edu.cn

Abstract. Optimal transportation is a fundamental topic that has attracted a great amount of attention from machine learning community in the past decades. In this paper, we consider an interesting discrete dynamic optimal transport problem: can we efficiently update the optimal transport plan when the weights or the locations of the data points change? This problem is naturally motivated by several applications in machine learning. For example, we often need to compute the optimal transportation cost between two different data sets; if some change happens to a few data points, should we re-compute the high complexity cost function or update the cost by some efficient dynamic data structure? We are aware that several dynamic maximum flow algorithms have been proposed before, however, the research on dynamic minimum cost flow problem is still quite limited, to the best of our knowledge. We propose a novel 2D Skip Orthogonal List together with some dynamic tree techniques. Although our algorithm is based on the conventional simplex method, it can efficiently complete each pivoting operation within $O(|V|)$ time with high probability where V is the set of all supply and demand nodes. Since dynamic modifications typically do not introduce significant changes, our algorithm requires only a few simplex iterations in practice. So our algorithm is more efficient than re-computing the optimal transportation cost that needs at least one traversal over all the $O(|E|) = O(|V|^2)$ variables in general cases. Our experiments demonstrate that our algorithm significantly outperforms existing algorithms in the dynamic scenarios.

1 Introduction

The discrete optimal transport problem involves finding the optimal transport plan "X" that minimizes the total cost of transporting one weighted dataset A to another B, given a cost function "C" [25]. The datasets A and B represent the supply and demand node sets, respectively, and the problem can be represented as a minimum cost flow problem by adding edges between A and B to create a complete bipartite graph. The discrete optimal transport problem finds numerous applications in areas such as image registration [15], seismic tomography [22], and machine learning [34]. However, these applications traditionally consider a static scenario where the weights of the datasets and the cost function remain constant. Yet, many real-world applications need to handle dynamic optimal transport problems:

- **Dataset Similarity**. In data analysis, measuring the similarity between datasets is a crucial task, and optimal transport has emerged as a powerful tool for this purpose [2]. Real-world datasets are often dynamic, with data points being replaced, weights adjusted, entries removed, or new data points added over time. Therefore, addressing the concept of dynamic optimal transport becomes essential.
- **Time Series Analysis**. Optimal Transport can serve as a metric in time series analysis [7]. The main intuition lies in the smooth transition of states between time points in a time series. The smoothness implies the potential to iteratively refine a new solution based on the previous one, circumventing the need for a complete recomputation.
- **Neuroimage analysis** [14, 18]. In the medical imaging applications, we may want to compute the change trend of a patient's organ (e.g., the MRI images of human brain over several months), and the differences are measured by the optimal transportation cost. Since the changes are often local and small, we can apply our method to quickly update the cost over the period.
- **Domain adaptation**. It has been shown that OT can be used to compute the transformation from one dataset (the source domain) to another dataset (the target domain) [12]. But the datasets may need to be updated over time (e.g., in social network the user's data can be changed), and so we can apply our method to quickly update the transformation between different domains.

Existing methods, such as the Sinkhorn algorithm [9] and the Network Simplex algorithm [23], are not adequately equipped to handle this dynamic problem. Upon any modification to the cost matrix, the Sinkhorn algorithm requires at least one regularization of the solution matrix, while the Network Simplex algorithm needs to traverse all edges at least once to ensure optimality. Consequently, the time complexity of these algorithms on the dynamic model is $\Omega(|V|^2)$.

Our novel algorithm and data structure present an efficient alternative, offering an $O(s|V|)$ solution for handling evolving datasets, where s is determined by the magnitude of the modification. In practice, s is much less than n, making our algorithm an invaluable tool for a wide range of applications in data science and beyond.

1.1 Related Works

Exact Solutions. Over the years, researchers have developed linear program minimum cost flow algorithms to address discrete optimal transport problems. The simplex method by Danzig et al. [10] offers a solution for general linear programs. Despite its worst-case exponential time complexity, Spileman and Teng [30] showed that its average time complexity is polynomial through smooth analysis. Cunningham [8] adapted the simplex method for minimum cost flow problems. Orlin [23] enhanced the network simplex algorithm with cost scaling techniques. Recently, Lee and Sidford's algorithm [20] presented an algorithm with a time complexity upper bound of $\tilde{O}(|E|\sqrt{|V|}\log^2 U)$, and Li et al. [21] proposed a parallel algorithm.

Approximations. Sherman [27] proposed a $(1+\epsilon)$ approximation algorithm. Pele and Werman [24] introduced the FastEMD algorithm that applies classic algorithms on a heuristic sketch of the input graph. Cuturi [9] used Sinkhorn-Knopp iterations to approximate the optimal transport problem by adding regular entropies to the optimization target. Recent optimizations on the Sinkhorn algorithm include Overrelaxing Sinkhorn by Thibault et al. [33] and the Screening Sinkhorn algorithm by Alaya et al. [1].

Dynamic Trees. Our data structure incorporates techniques from dynamic trees, which maintain and manipulate tree-like structures under specific operations. Sleater and Tarjan [28] proposed the Link/Cut tree data structure for path aggregate maintenance. Tarjan and Vishkin [32] introduced the Euler Tour Technique for maintaining a tree as a cyclic ordered set of its Euler Tour sequence. Tarjan [31] later used splay trees to maintain such information, leading to the development of Euler Tour Trees.

Search Trees. Our data structure utilizes high-dimensional extensions of skip lists to maintain a 2-dimensional Euler Tour sequence. Existing high-dimensional data structures based on self-balanced binary search trees, such as Bentley's k-d tree [3], are not suitable as they do not support cyclic ordered set maintenance. Pugh's concept of skip lists [26], which are linked lists with additional layers of pointers for element skipping, is adapted in our context to form skip orthogonal lists.

1.2 Overview of Our Algorithm

Our algorithm for the dynamic optimal transportation problem employs two key strategies:

First, dynamic optimal transport operations are reduced to simplex iterations. Our technique, grounded on the Simplex method, operates by eliminating the smallest cycle in the graph. We assume that modifications influence only a small portion of the result, requiring only a few simplex iterations. However, existing algorithms like the Network Simplex Algorithm perform poorly under dynamic modifications as they require scanning all edges at least once to ensure global minimum attainment.

Second, a data structure is provided for performing each simplex iteration within expected linear time relative to the supply and demand set. Our data structure, described in Section 3, employs the Euler Tour Technique. We adapt skip lists to maintain the cyclic ordered set produced by the Euler Tour Technique and introduce an additional dimension to create a Skip Orthogonal List. This structure aids in maintaining information about $A \times B \mapsto \mathbb{R}$, crucial for our pivoting step.

The rest of the paper is organized as follows:

- Section 3 will give a introduction on the data structure *Skip Orthogonal List* we innovated, where subsection 3.1 explains how the data structure is organized and subsection 3.2 uses *cut* operation as an example to demonstrate one of the updates on this data structure.

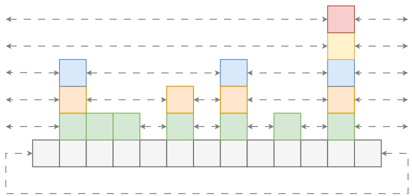

Fig. 1. 1D Euler Tour Tree with skip list

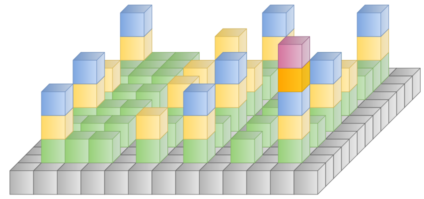

Fig. 2. 2D Euler Tour Tree with Skip Our Orthogonal List

- Section 4 explains how to use our data structure to solve the dynamic optimal transport problem, by using subsection 4.1 to show that dynamic optimal transport model could be reduced to simplex iterations, while using subsection 4.2 to show how our data structure could be used to improve the performance of each simplex iteration.

2 Preliminaries

2.1 Optimal Transport

Let A and B represent source and target point sets, respectively, with discrete probability distributions $\alpha : A \mapsto \mathbb{R}_{\geq 0}$ and $\beta : B \mapsto \mathbb{R}_{\geq 0}$, such that $\sum_{a\in A} \alpha_a = \sum_{b\in B} \beta_b = 1$. The cost function $C : A \times B \mapsto \mathbb{R}$ includes entries c_{ab} denoting the cost of transporting a unit of mass from point $a \in A$ to point $b \in B$. The discrete optimal transport problem can be formulated as (1).

$$
\begin{aligned}
W(\alpha, \beta, C) &\triangleq \min_{X: A\times B \mapsto \mathbb{R}_{\geq 0}} \sum_{a\in A}\sum_{b\in B} c_{ab}x_{ab} \\
\text{subject to} &\quad \begin{cases} \sum_{b\in B} x_{ab} = \alpha_a & \forall a \in A \\ \sum_{a\in A} x_{ab} = \beta_b & \forall b \in B \end{cases}
\end{aligned}
\tag{1}
$$

Since Problem (1) is a network flow problem, it can be transformed into Problem (2) by adding infinity-cost edges, as constraint (3) indicates, and redefining point weights as (4) [25]. Note that in (2), the input weight w must always satisfy $\sum_{v\in V} w_v = 0$. Otherwise, the constraints cannot be satisfied.

$$
\begin{aligned}
W(w, C) &\triangleq \min_{X: V\times V \mapsto \mathbb{R}_{\geq 0}} \sum_{u\in V}\sum_{v\in B} c_{uv}x_{uv} \\
\text{subject to} &\quad \sum_{v\in V} x_{uv} - \sum_{v\in V} x_{vu} = w_u \quad \forall u \in V
\end{aligned}
\tag{2}
$$

$$
c_{aa} = c_{ba} = c_{bb} = +\infty \quad \forall a \in A, b \in B \tag{3}
$$

$$
w_u = \begin{cases} \alpha_a & \text{if } u = a \in A \\ -\beta_b & \text{if } u = b \in B \end{cases} \tag{4}
$$

For a basic transportation plan X, we notice that the basis variables always form a spanning tree of the complete directed graph with self loops $G(V, V^2)$ [8]. Let the dual variables be $\pi : V \mapsto \mathbb{R}$, satisfying constraint (5). We may then define the adjusted cost function $C^\pi : V^2 \mapsto \mathbb{R}$ as $C^\pi_{uv} \triangleq C_{uv} + \pi_v - \pi_u$, where C^π_{uv} represents the simplex multipliers for the linear program [23].

$$x_{uv} \text{ is a basic variable } \implies \pi_u - \pi_v = C_{uv} \tag{5}$$

2.2 Euler Tour Technique

The Euler Tour Technique is a method for representing a tree $T(V, E)$ as a cyclic ordered set of length $|V|+2|E| = O(|V|)$ [31]. Given a tree $T(V, E)$, we construct a directed graph $D_T(V, E')$ as follows:

- For each vertex $v \in V$, add the self-loop (v, v) to E'.
- For each undirected edge $\{u, v\} \in E$, add two directed edges (u, v) and (v, u) to E'.

Since the difference of In-Degree and Out-Degree of each vertex in D_T is 0, D_T must contain an Euler Tour;

Definition 1. *Define the Euler Tour representation of tree T as an arbitrary sequence of Euler Tour of D_T represented by edges. Denote $E(D_T)$ as the circular ordered set with regard to the Euler Tour.*

Through Definition 1, we can reduce *edge linking*, *edge cutting*, *sub-tree weight modification* and *sub-tree weight querying* to constant number of element insertion, element deletion, range weight modification and range weight querying on a circular ordered set [31]. We show in Section 2.1 that the dynamic optimal transport could be reduced to the 2 dimensional version of these four operations, where V would be interpreted as the union of supply node set and demand node set (i.e. dual variables of the OT problem), and E would be explained as the set of edges that are potentially non-zero, i.e., a feasible basis for the primal variables. Euler Tour Technique is crucial to our work.

2.3 Orthogonal Lists and Skip Lists

Skip Lists are probabilistic data structures that extend a singly linked list with forward links at various levels, improving search, insertion, and deletion operations. Figure 1 demonstrates a skip list. Each level contains a circular linked list, where lists at a higher level is a subset of lists at a lower level and the bottom level contains all elements. Nodes on the same level is painted by the same color and linked horizontally. Corresponding elements in adjacent lists are connected by vertical pointers. We apply this skipping technique to circular singly linked lists in our work.

Most self-balanced binary search trees support lazy propagation techniques, allowing range modifications within $O(\log n)$ time, where n is the sequence length maintained by the tree [29]. This technique is commonly used in dynamic trees for network problems [31].

A k-dimensional Orthogonal List has k orthogonal forward links and reduces to a standard linked list when $k = 1$. Orthogonal lists, which can be singly, doubly, or circularly linked, can maintain information mapped from the Cartesian product of ordered sets, such as sparse tensors [6].

Figure 1 demonstrates an orthogonal list maintaining a 3×4 matrix. Each node has 2 forward links, denoted by *row* links (yellow) and *column* links (green). *Row* links connect elements in each row into a circular linked list horizontally and *column* links connect elements in each column into a circular linked list vertically.

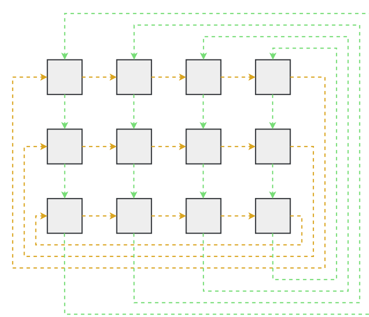

Fig. 3. An Orthogonal List Example

3 Skip Orthogonal List

This section introduces our novel data structure **Skip Orthogonal List**. Section 4 uses this data structure for fast update in the dynamical optimal transport problem.

Formally, with the help of a Skip Orthogonal List, we can maintain a forest $T(V, E)$ with at most two trees, and a function $C^{\pi} : V^2 \mapsto \mathbb{R}$, supporting the following operations:

- **Cut**. Given $\{u, v\}$, remove edge $\{u, v\}$ from the forest, splitting a tree into two disjoint trees, provided that the forest consists of exactly one tree and $\{u, v\} \in E$. Let the connected component containing u form the vertex set V_1, and the connected component containing v form the vertex set V_2.
- **Insert**. Add a new node to the forest, provided that the forest consists of exactly one tree. Let the original nodes form the vertex set V_1, and the new node itself form the vertex set V_2.

- **Range Add**. Given x, for each $(u,v) \in V^2$, update c^π_{uv} as equation (6), provided the forest contains exactly two trees.

$$c^\pi_{uv} \leftarrow c^\pi_{uv} + \begin{cases} 0 & (u,v) \in V_1 \times V_1 \\ -x & (u,v) \in V_1 \times V_2 \\ x & (u,v) \in V_2 \times V_1 \\ 0 & (u,v) \in V_2 \times V_2 \end{cases}, \tag{6}$$

- **Link**. Given (u,v), add edge (u,v) to the forest, connecting two disjoint trees into a single tree, provided that u and v are disconnected.
- **Global Minimum Query**. Return the minimum value of C^π on the tree, provided that the forest is connected.

For the remaining of the section, we will construct a data structure in expected $O(|V|^2)$ amount of space where each operation can be done within expected $O(|V|)$ amount of time. Section 3.1 shows the overall structure of the data structure and how to query this data structure. Section 3.2 will show the cut operation based on this structure. For other operations (linking, insertion, range adding), we have put it in the appendix for due to the limited space.

3.1 Overall Structure

As Figure 4 shows, a **Skip Orthogonal List** is a layered collection of Orthogonal lists, just like the fact that a skip list is a hierarchical view of a list. Each layer has fewer elements than the one below it, and the elements are evenly spaced out. The bottom layer has all the elements while the top layer has the least. Formally, it can be defined as Definition 2.

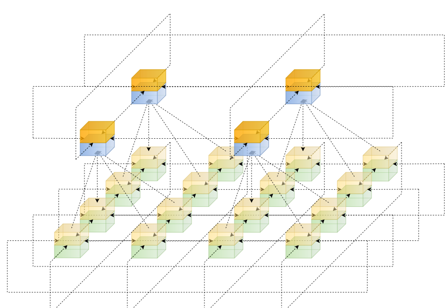

Fig. 4. Detailed Structure of Skip Orthogonal List

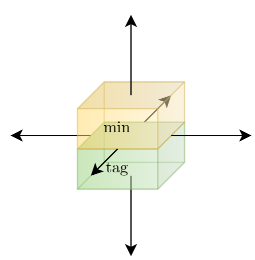

Fig. 5. Node Structure of Skip Orthogonal List

Definition 2. *Given a parameter p and a cyclic ordered set S, a* ***2 Dimensional Skip Orthogonal List*** *is an infinite collection of 2 Dimensional Circular Orthogonal Lists $\mathcal{L} = (L_0, L_1, \cdots)$, where*

– $\{h_s : s \in S\}$ *are* $|S|$ *independent random variable. The distribution is a geometric distribution with parameter* p
– *For each* $i \in \mathbb{N}$*, let* L_i *maintain all elements in* S_i^2*, where* S_i *is the cyclic sequence formed by* $\{s \in S \mid h_s \geq i\}$

Assume we use this data structure to maintain several information about $E(D_T) \times E(D_T)$. Since $|E(D_T)| = O(|V|)$ as discussed in Section 3, from Definition 2, the expected space complexity is $O(|V|^2)$ (due to the space limit, we put our proof in appendix).

Now we augment this data structure to store some additional information for range adding and global minimum query. We first adapt the definition of "dominate" to the 2 dimensional case defined as Definition 3 . In Figure 6 and Figure 7, the blue nodes dominate itself and all yellow nodes, while the red nodes dominate every node in the orthogonal list.

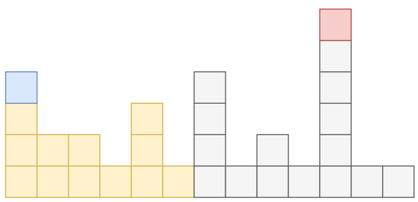

Fig. 6. Domination in Skip List

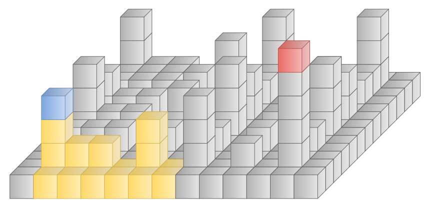

Fig. 7. Domination in 2D Skip List

Definition 3. *In a Skip Orthogonal List* $\mathcal{L}$*,* $x_{i_1 i_2}$ ***dominates*** $x_{j_1 j_2}$ *at level* l *if and only if* ($x_{i_1 i_2} = x_{j_1 j_2}$ *or the following 3 conditions are all satisfied:*

– $h_{x_{i_1}} \geq l$ *and* $h_{x_{i_2}} \geq l$
– $h_{x_{i_1+1}} < l$ *and* $h_{x_{i_1+2}} < l$ *and ... and* $h_{x_{j_1-1}} < l$ *and* $h_{x_{j_1}} < l$
– $h_{x_{i_2+1}} < l$ *and* $h_{x_{i_2+2}} < l$ *and ... and* $h_{x_{j_2-1}} < l$ *and* $h_{x_{j_2}} < l$

Similar techniques can be used to define Skip Orthogonal Lists of higher dimensions, but for dynamic optimal transport operations, since we now only need to maintain the adjusted cost function C^π, we only need a 2 dimensional instance for the circular ordered set $E(D_T)$. We use $x_{\tilde{u}\tilde{v}}$ to maintain c_{uv}^π where $\tilde{u}$ and $\tilde{v}$ denote the self loop (u, u) and (v, v) respectively and fill the remaining entries of the Skip Orthogonal Lists as the identity for the global minimum query, i.e. $+\infty$.

We now augment the Skip Orthogonal List defined in 2. For each node $x_{i_1 i_2}$ at orthogonal list L_l, beside the 2 forward links and 2 backward links, we add the following attributes as shown in Figure 5:

– *min* to maintain the minimum amongst all nodes dominated by it if $l > 0$
– *tag* to maintain the lazy tag amongst for all nodes dominated by it if $l > 0$
– *son* to point to $x_{i_1 i_2}$ at orthogonal list L_{l-1} if $l > 0$
– *parent* to point to the node that dominates it if L_{l+1} is not empty

3.2 Update (Cut)

In this subsection, we focus on the Update (Cut) algorithm for a 2D Skip Orthogonal List as an example. Figure 8 and Figure 9 show the basic idea of the cutting process.

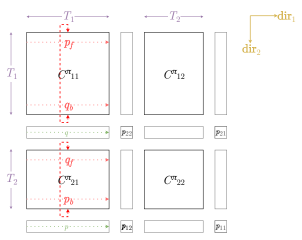

Fig. 8. Vertical View of Update (Cut)

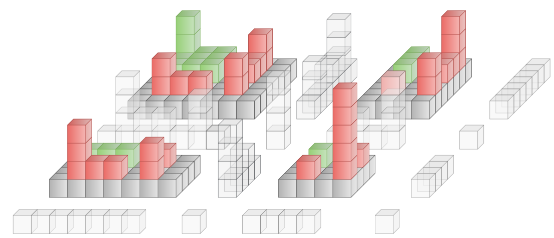

Fig. 9. 3D View of Update (Cut)

The algorithm can be crudely described as follows:

1. Find the 2 rows and 2 columns representing directed edge (u, v) and (v, u) in $E(D_T)$, i.e. the transparent nodes in Figure 9 or the gray nodes in Figure 8.
2. Push down the *tag* attribute of all nodes alongside the rows and columns, i.e. the red nodes and transparent nodes in Figure 9. Only the nodes of which the set of nodes dominated by them may change during step 3 need to push down.
3. Cut the rows and columns, warping up the forward links and backward links of points alongside, as Figure 8 demonstrates. This will cut the original Skip Orthogonal List into 4 small lists.
4. Update the *min* attribute of all nodes whose descendants were affected in step 3, i.e. the red nodes in Figure 9.
5. Return the 4 small lists cutted out in step 3.

From the discussion above, we can see that this algorithm is an augmentation of *lazy propagation technique* which is widely used in normal balanced tree, and thus solves our problem. We can see that the time complexity is linear corresponding to the number of nodes dominated by the red nodes and transparent nodes in figure 9. Similar to the proof of space complexity, the expected time complexity is $O(|V|)$ with high probability.

4 Our Dynamic Network Simplex Method

The simplex method performs simplex iterations on some initial feasible basis until optimality is obtained [10]. Simplex iterations can refine the current solution. In each simplex iteration, some variable with negative simplex multiplier is selected by some strategy to perform a *pivot* subroutine. One possible strategy is to pivot in the variable with the smallest simplex multiplier. Section 4.1 focuses on defining dynamic optimal transport operations and using simplex iterations to solve it, while Section 4.2 focuses on analyzing the details in each simplex iteration to optimize it.

4.1 Dynamic Optimal Transport Operations

In an Optimal Transport problem, suppose the supply set A and the demand set B are mapped to some points in some metric space $(\mathcal{X}; d)$ by function f, e.g. X could be the Euclidean Space $\mathbb{R}^k$ while d could be the Euclidean Distance ℓ_2, while the cost function C_{ab} is usually defined as $d(f_a, f_b)$. Let $w : V \mapsto \mathbb{R}$ denote the weight function as equation (4) defined where $V = A \cup B$. A **Dynamic Optimal Transport** algorithm supports the following 4 types of update and 1 type of online query:

- **Space Position Modification.** Select some supply or demand point $v \in V$ and move f_v to another point $x \in \mathcal{X}$. This usually results in modification of an entire row or column in the cost matrix $C \in \mathbb{R}^{A \times B}$.
- **Weight Modification.** Select a pair of supply or demand point $u, v \in V$ with some positive number $\Delta \in \mathbb{R}_+$. Let $w_u \leftarrow w_u - \Delta$ while $w_v \leftarrow w_v + \Delta$.
- **Point Deletion.** Select $v \in V$ with $w_v = 0$. Remove v from V.
- **Point Insertion.** Select $v \notin V$ and $f_v \in \mathcal{X}$. Let $w_v = 0$ and insert v into set V
- **Query.** Answer the current Optimal Transport plan and value.

These updates do not modify the sum of all points in supply and demand set V, so $\sum_{v \in V} w_v \equiv 0$ and thus a transport plan always exists. Therefore we can reduce these updates to updates on simplex basis:

- **Space Position Modification.** The original optimal solution is primal feasible but not primal optimal, i.e. not dual feasible. We perform primal simplex method based on the original optimal solution. When moving point v, we first update cost matrix C, dual variable π and modified cost C^π to meet constraint (5). After that, we perform simplex iterations while the minimum value of the adjusted cost C^π is negative [25].
- **Weight Modification.** The original optimal solution is dual feasible but not primal feasible. We perform dual simplex iterations based on the original optimal solution. Assume we try to decrease w_u and increase w_v by Δ, we send Δ amount of flow from u to v on the residual network similar to the shortest path augmenting method [11, 25]: We send flow on basic variables X_B. If some variable needs to be pivoted out, we pivot in the variable with the smallest adjusted cost.
- **Point Deletion & Point Insertion.** As the deleted/inserted point weighs 0, whether inserting or deleting the point does not influence our result. In order to influence the Optimal Transport plan, it must be combined with several **Weight Modification** operations for weight assignment. Therefore,

we can maintain a node pool to maintain all supply and demand nodes with 0 weight. Each **Point Insertion** operation takes some point in this pool and move it to the correct place while each **Point Deletion** operation returns a node to the pool.

Our solution updates the optimal transport plan as soon as an update takes place, so we can answer the online query about optimal transport plan and value. Since we are only modifying one or two nodes in the dataset, intuitively the optimal transport plan should not change much, and we do not need many simplex iterations to obtain optimality. Assume we need s simplex iterations, where we assume $s \ll |V|$. Therefore the time complexity for our algorithm is $O(s\text{Time}_{\text{iteration}})$ where $\text{Time}_{\text{iteration}}$ is the time of each simplex iteration.

4.2 Simplex Iteration Details

As Section 4.1 describes, dynamic operations on Optimal Transport could be reduced to simplex iterations. In this section, we review on the operations used in conventional network simplex algorithm and use data structure designed in Section 3 to maintain C^π.

Conventional network simplex method is simplex method simplified by some graph properties [8]. The steps of a (network) simplex iteration can be described as:

1. **Select Variable to Pivot in.** Select variables with the smallest adjusted cost C^π to pivot in. Assume $x_{i_{\text{in}}j_{\text{in}}}$ is to be pivoted in.
2. **Update Primal Solution.** Send circular flow in the cycle formed by adding the new variable to the current basis to perform the pivot operation until some basic variable is pivoted out, i.e. some basic variable in the reverse direction runs out of flow. Assume $x_{i_{\text{out}}j_{\text{out}}}$ is to be pivoted out
3. **Update Dual Solution.** Update dual variables π and modified cost C^π to meet constraint (5), as the new basis as C^π will soon be queried in the next simplex iteration.

The selecting step performs a query on the data structure on C^π about the minimum element and the dual updating performs an update. Though the primal updating step can be done within $O(\log |V|)$ amount of time [31], conventional network simplex maintain C^π through brute force. Upon an query, conventional network simplex brutally traverse through all adjusted costs and selects the minimum, while upon an update, conventional network simplex updates the adjusted cost one by one after the adjusted cost is computed. This indicates that the time complexity of each simplex iteration is $O(|V|^2)$.

Our goal is to maintain C^π so that it can answer the global minimum query and perform update when the primal basis changes. In a simplex method, when we decide to pivot in the variable $x_{i_{\text{in}}j_{\text{in}}}$, we update the dual variables as equation (7) describes where V_1 is the set of nodes connected to i_{in} and V_2 is the set of nodes connected to j_{in} after the edge $x_{i_{\text{out}}j_{\text{out}}}$ is cut [8]. π are the dual variables before pivoting and π' are the dual variables after pivoting.

$$\pi_i' \leftarrow \pi_i + \begin{cases} C^{\pi}_{i_{\text{in}}j_{\text{in}}} & i \in V_1 \\ 0 & i \in V_2 \end{cases} \tag{7}$$

From the definition of the adjusted cost function C^{π}, we obtain that the update objective can be formulated as (8) where C^{π} is the adjusted cost function with regard to the old dual variables π while $C^{\pi'}$ regards the new dual variables π'. Therefore, Algorithm 1 can help us with the adjusted cost function update. Our Skip Orthogonal List in Section 3 is capable of performing *cut*, *add* and *link* operation within $O(|V|)$ amount of time. Therefore Theorem 1 holds.

$$C^{\pi'}_{ij} = C^{\pi}_{ij} + \pi_i' - \pi_j' = C^{\pi}_{ij} + \begin{cases} 0 & (i,j) \in V_1 \times V_1 \\ -C^{\pi}_{i_{\text{in}}j_{\text{in}}} & (i,j) \in V_1 \times V_2 \\ C^{\pi}_{i_{\text{in}}j_{\text{in}}} & (i,j) \in V_2 \times V_1 \\ 0 & (i,j) \in V_2 \times V_2 \end{cases} \tag{8}$$

Theorem 1. *Each simplex iteration in conventional network simplex can be completed within expected $O(|V|)$ amount of time.*

Algorithm 1: Adjusted Cost Matrix Update

Input: Adjusted cost function C^{π}, entering variable $x_{i_{\text{in}}j_{\text{in}}}$, leaving variable $x_{i_{\text{out}}j_{\text{out}}}$

Output: Updated adjusted cost function $C^{\pi'}$

$t \leftarrow C^{\pi}_{a_{\text{in}}b_{\text{in}}}$

$C^{\pi}_{11}, C^{\pi}_{12}, C^{\pi}_{21}, C^{\pi}_{22} \leftarrow \mathit{cut}(C^{\pi}, a_{\text{in}}, b_{\text{out}})$

$\mathit{add}(C^{\pi}_{12}, -t)$

$\mathit{add}(C^{\pi}_{21}, t)$

$C^{\pi'} \leftarrow \mathit{link}(C^{\pi}_{11}, C^{\pi}_{12}, C^{\pi}_{21}, C^{\pi}_{22}, a_{\text{in}}, b_{\text{in}})$

5 Experiments

All the results were obtained on a server equipped with *AMD Ryzen Threadripper PRO 5975WX 32-Cores CPU* and *512GB* main memory with frequency *3200 MHz*; the data structures are implemented in *C++20* compiled by *G++ 13.1.0* on *Ubuntu 22.04.3*. The data structures are compiled to shared objects to be called by *Python 3.11.5*. Our code uses Network Simplex library from [5] to get an initial feasible flow and *NumPy* [16] to represent points in the space.

In our experiment, we used **Network Simplex** algorithm [23] and **Sinkhorn** algorithm [9] from the Python Optimal Transport (POT) library [13]. We run our algorithm in both **Space Position Modification** and **Point Insertion** scenario as described in Section 4.1. We use the running time to measure the performance of algorithms.

Datasets. We study the performance of our algorithm on both synthetic datasets and real datasets. For synthetic datasets, we construct a mixture of 2 Gaussian distribution in $\mathbb{R}^2$, where points in the same Gaussian distribution share the same label [4]. We also used MNIST dataset as a popular real world dataset [19]. We partition the labels into 2 groups and run optimal transport between these groups, as we wish to run optimal transport on large datasets, which indicates that a single label cannot meet the number of nodes we demand.

Setup. We set the Sinkhorn regularization parameter λ as 0.1 and scale the median of the cost matrix C to be 1. We test on sampled datasets where we vary the the node size $|V|$ up to 4×10^4. For each dataset, we test the static running time of POT on our machine and executed each dynamic operations 100 times to calculate the means and standard errors of our algorithm. For *Space Position Modification*, we randomly choose some point and add a random Gaussian noise with a variance of 0.5 in each dimension to it. For *Point Insertion*, we randomly select a point in the dataset that is not currently in the OT system to insert. We perform a query after these updates and compare with static algorithms implemented in POT to ensure our correctness.

Result and Analysis. We illustrate our experiment results in Figure 10 with the help of Matplotlib [17]. In dynamic models, our algorithm is about 1000 times faster than Static Network Simplex algorithm and 10 times faster than Sinkhorn Algorithm when $|V|$ reaches 40000 depending on the amplitude of modification. As the number of nodes grow larger, our algorithm will be even faster. This indicates that our algorithm is fast in the case where simplex iterations are not much, as discussed in Section 4.1. Also, the running time of our algorithm is in a super linear trend. As each simplex iteration in our method is strictly linear in expectation theoretically, this could be caused by increment in number of simplex iterations or decrement in cache hit rate as the node size grows larger.

6 Conclusion and Future Work

Our work provides a data structure which is useful in traditional network simplex method. With the help of our data structure, the time complexity of the whole pivoting process is $O(|V|)$ in expectation as Theorem 1 describes. Based on the fact that conventional simplex method performs well in average cases, our algorithm has a better theoretical performance than most existing algorithms.

However, our algorithm lead to several performance issues in practice. First, as our algorithm stores the entire 2D Skip Orthogonal List data structure, our algorithm require additional space. Second, as our algorithm is based on linked data structures, the cache hit rate is not high. Therefore, improving our implementation and developing algorithms and data structures with similar complexity but more memory friendly is our next goal.

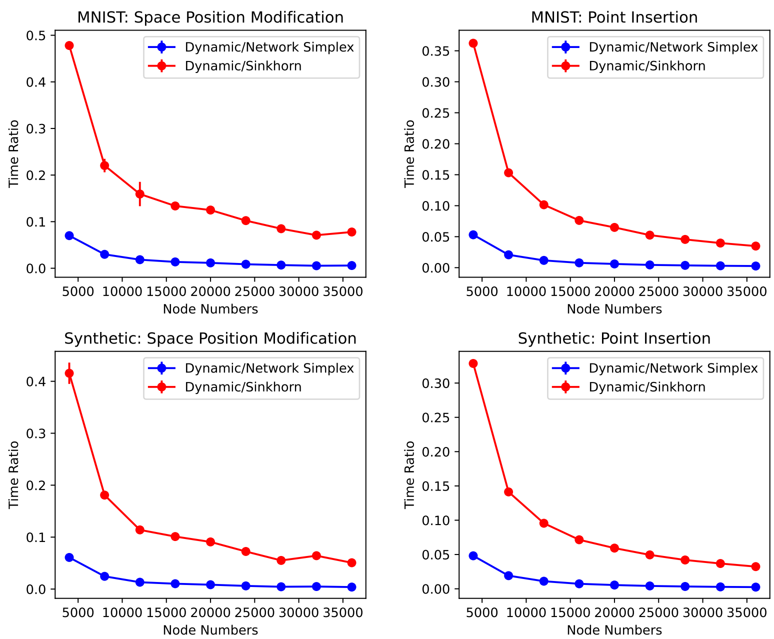

Fig. 10. Experiment Result

Bibliography

[1] Alaya, M.Z., Berar, M., Gasso, G., Rakotomamonjy, A.: Screening sinkhorn algorithm for regularized optimal transport. Advances in Neural Information Processing Systems **32** (2019)

[2] Alvarez-Melis, D., Fusi, N.: Geometric dataset distances via optimal transport. In: NeurIPS 2020. ACM (February 2020), `https://www.microsoft.com/en-us/research/publication/geometric-dataset-distances-via-optimal-transport/`

[3] Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM **18**(9), 509–517 (1975)

[4] Blum, A., Hopcroft, J., Kannan, R.: Foundations of data science. Cambridge University Press (2020)

[5] Bonneel, N., van de Panne, M., Paris, S., Heidrich, W.: Displacement Interpolation Using Lagrangian Mass Transport. ACM Transactions on Graphics (SIGGRAPH ASIA 2011) **30**(6) (2011)

[6] Butterfield, A., Ngondi, G.E., Kerr, A.: A dictionary of computer science. Oxford University Press (2016)

[7] Cheng, K., Aeron, S., Hughes, M.C., Miller, E.L.: Dynamical wasserstein barycenters for time-series modeling. Advances in Neural Information Processing Systems **34**, 27991–28003 (2021)

[8] Cunningham, W.H.: A network simplex method. Mathematical Programming **11**, 105–116 (1976)

[9] Cuturi, M.: Sinkhorn distances: Lightspeed computation of optimal transport. Advances in neural information processing systems **26** (2013)

[10] Dantzig, G.B., Orden, A., Wolfe, P., et al.: The generalized simplex method for minimizing a linear form under linear inequality restraints. Pacific Journal of Mathematics **5**(2), 183–195 (1955)

[11] Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM (JACM) **19**(2), 248–264 (1972)

[12] Flamary, R., Courty, N., Tuia, D., Rakotomamonjy, A.: Optimal transport for domain adaptation. IEEE Trans. Pattern Anal. Mach. Intell **1**, 1–40 (2016)

[13] Flamary, R., Courty, N., Gramfort, A., Alaya, M.Z., Boisbunon, A., Chambon, S., Chapel, L., Corenflos, A., Fatras, K., Fournier, N., et al.: Pot: Python optimal transport. The Journal of Machine Learning Research **22**(1), 3571–3578 (2021)

[14] Gramfort, A., Peyré, G., Cuturi, M.: Fast optimal transport averaging of neuroimaging data. In: Information Processing in Medical Imaging: 24th International Conference, IPMI 2015, Sabhal Mor Ostaig, Isle of Skye, UK, June 28-July 3, 2015, Proceedings 24. pp. 261–272. Springer (2015)

[15] Haker, S., Zhu, L., Tannenbaum, A., Angenent, S.: Optimal mass transport for registration and warping. International Journal of computer vision **60**, 225–240 (2004)

[16] Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E.: Array programming with NumPy. Nature **585**(7825), 357–362 (Sep 2020). https://doi.org/10.1038/s41586-020-2649-2, `https://doi.org/10.1038/s41586-020-2649-2`

[17] Hunter, J.D.: Matplotlib: A 2d graphics environment. Computing in Science & Engineering **9**(3), 90–95 (2007). https://doi.org/10.1109/MCSE.2007.55

[18] Janati, H., Bazeille, T., Thirion, B., Cuturi, M., Gramfort, A.: Group level meg/eeg source imaging via optimal transport: minimum wasserstein estimates. In: Information Processing in Medical Imaging: 26th International Conference, IPMI 2019, Hong Kong, China, June 2–7, 2019, Proceedings 26. pp. 743–754. Springer (2019)

[19] LeCun, Y., Cortes, C., Burges, C.: Mnist handwritten digit database. ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist **2** (2010)

[20] Lee, Y.T., Sidford, A.: Path finding methods for linear programming: Solving linear programs in o (vrank) iterations and faster algorithms for maximum flow. In: 2014 IEEE 55th Annual Symposium on Foundations of Computer Science. pp. 424–433. IEEE (2014)

[21] Li, W., Ryu, E.K., Osher, S., Yin, W., Gangbo, W.: A parallel method for earth mover's distance. Journal of Scientific Computing **75**(1), 182–197 (2018)

[22] Métivier, L., Brossier, R., Mérigot, Q., Oudet, E., Virieux, J.: Measuring the misfit between seismograms using an optimal transport distance: Application to full waveform inversion. Geophysical Supplements to the Monthly Notices of the Royal Astronomical Society **205**(1), 345–377 (2016)

[23] Orlin, J.B.: A polynomial time primal network simplex algorithm for minimum cost flows. Mathematical Programming **78**, 109–129 (1997)

[24] Pele, O., Werman, M.: Fast and robust earth mover's distances. In: 2009 IEEE 12th international conference on computer vision. pp. 460–467. IEEE (2009)

[25] Peyré, G., Cuturi, M., et al.: Computational optimal transport: With applications to data science. Foundations and Trends® in Machine Learning **11**(5-6), 355–607 (2019)

[26] Pugh, W.: Skip lists: a probabilistic alternative to balanced trees. Communications of the ACM **33**(6), 668–676 (1990)

[27] Sherman, J.: Generalized preconditioning and undirected minimum-cost flow. In: Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 772–780. SIAM (2017)

[28] Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. In: Proceedings of the thirteenth annual ACM symposium on Theory of computing. pp. 114–122 (1981)

[29] Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. Journal of the ACM (JACM) **32**(3), 652–686 (1985)

[30] Spielman, D.A., Teng, S.H.: Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. Journal of the ACM (JACM) **51**(3), 385–463 (2004)

[31] Tarjan, R.E.: Dynamic trees as search trees via euler tours, applied to the network simplex algorithm. Mathematical Programming **78**(2), 169–177 (1997)

[32] Tarjan, R.E., Vishkin, U.: Finding biconnected componemts and computing tree functions in logarithmic parallel time. In: 25th Annual Symposium onFoundations of Computer Science, 1984. pp. 12–20. IEEE (1984)

[33] Thibault, A., Chizat, L., Dossal, C., Papadakis, N.: Overrelaxed sinkhorn–knopp algorithm for regularized optimal transport. Algorithms **14**(5), 143 (2021)

[34] Torres, L.C., Pereira, L.M., Amini, M.H.: A survey on optimal transport for machine learning: Theory and applications. arXiv preprint arXiv:2106.01963 (2021)