A Novel Skip Orthogonal List for Dynamic Optimal Transport Problem

Xiaoyang Xu, Hu Ding*

School of Computer Science and Technology
University of Science and Technology of China
xiaoyangxu@mail.ustc.edu.cn, huding@ustc.edu.cn

Abstract

Optimal transport is a fundamental topic that has attracted a great amount of attention from the optimization 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 transport cost between two different data sets; if some changes happen 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 find the variable to pivot within expected $O(1)$ time, and complete each pivoting operation within expected $O(|V|)$ time 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 transport cost that needs at least one traversal over all $|E| = O(|V|^2)$ variables, where $|E|$ denotes the number of edges in the network. Our experiments demonstrate that our algorithm significantly outperforms existing algorithms in the dynamic scenarios.

Introduction

The discrete *optimal transport (OT)* 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 matrix "C" (Peyré, Cuturi et al. 2019). The datasets A and B respectively represent the supply and demand node sets, and the problem can be represented as a minimum cost flow problem by adding the edges between A and B to create a biclique. The discrete optimal transport problem finds numerous applications in the areas such as image registration (Haker et al. 2004), seismic tomography (Métivier et al. 2016), and machine learning (Torres, Pereira, and Amini 2021). However, most of these applications only consider static scenario where the weights of the datasets and the cost matrix remain constant. Yet, many real-world applications need to consider the dynamic scenarios:

- **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 (Alvarez-Melis and Fusi 2020). Real-world datasets are often dynamic, with data points being replaced, weights adjusted, or new data points added over time. Therefore, it is necessary to take these dynamically changes into account.
- **Time Series Analysis**. Optimal transport can serve as a metric in time series analysis (Cheng et al. 2021). 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** (Gramfort, Peyré, and Cuturi 2015; Janati et al. 2019). 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 transport cost. Since the changes are often local and small, we may hope to apply some efficient method to quickly update the cost over the period.

Denote by V and E the sets of vertices and edges in the bipartite network, respectively. Existing methods, such as the Sinkhorn algorithm (Cuturi 2013) and the Network Simplex algorithm (Orlin 1997), are not adequate to handle the dynamic scenarios. Upon any modification to the cost matrix, the Sinkhorn algorithm requires at least one Sinkhorn-Knopp iteration to regularize the entire the solution matrix, while the Network Simplex algorithm needs to traverse all edges at least once. Consequently, the time complexities of these algorithms for the dynamic model are $\Omega(|V|^2)$ in general cases.

Our algorithm takes a novel data structure that yields an $O(s|V|)$ time solution for handling evolving datasets, where s is determined by the magnitude of the modification. In

*Corresponding Author

practice, s is usually much less than the data size $|V|$, and therefore our algorithm can save a large amount of execution time for the dynamic scenarios. [1]

Related Works

Exact Solutions. In the past years, several linear programming based minimum cost flow algorithms have been proposed to address discrete optimal transport problems. The simplex method by Dantzig et al. (1955) can efficiently solve general linear programs. Despite its worst-case exponential time complexity, Spielman and Teng (2004) showed that its smoothed time complexity is polynomial. Cunningham (1976) adapted the simplex method for minimum cost flow problems. Further, Orlin (1997) enhanced the network simplex algorithm with cost scaling techniques and Tarjan (1997) improved its complexity to be $\tilde{O}(|V||E|)$. Recently, Van Den Brand et al. (2021) presented an algorithm based on the interior point method with a time complexity $O\left(|E|+|V|^{1.5}\right)$, and Chen et al. (2022) proposed a $|E|^{1+o(1)}$ time algorithm through a specially designed data structure on the interior point method.

Approximate Algorithms. For approximate optimal transport, Sherman (2017) proposed a $(1+\epsilon)$ approximation algorithm in $\epsilon^{-2}|E|^{1+o(1)}$ time. Pele and Werman (2009) introduced the FastEMD algorithm that applies classic algorithms on a heuristic sketch of the input graph. Later, Cuturi (2013) used Sinkhorn-Knopp iterations to approximate the optimal transport problem by adding the smoothed entropic entry as the regularization term. Recently several optimizations on the Sinkhorn algorithm have been proposed, such as the Overrelaxation of Sinkhorn (Thibault et al. 2021) and the Screening Sinkhorn algorithm (Alaya et al. 2019).

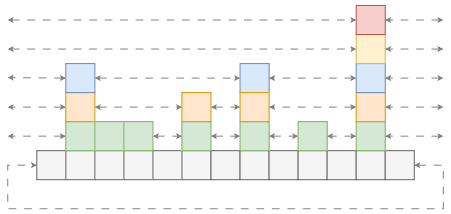
Search Trees and Skip Lists. Our data structure also utilizes high-dimensional extensions of skip lists to maintain a 2-dimensional Euler Tour sequence. Existing high-dimensional data structures based on self-balanced binary search trees, such as k-d tree (Bentley 1975), are not suitable as they do not support cyclic ordered set maintenance. Skip lists (Pugh 1990) as depicted in Figure 1a, which are linked lists with additional layers of pointers for element skipping, is adapted in our context to form skip orthogonal lists. This skipping technique is later generalized for sparse data in higher dimension (Nickerson 1994; Eppstein, Goodrich, and Sun 2005), but range querying generally requires $O(n^2)$ time where n^2 is the number of points in the high dimensional space. On the other hand, our data structure requires expected $O(n)$ time when applied to simplex iterations.

Overview of Our Algorithm

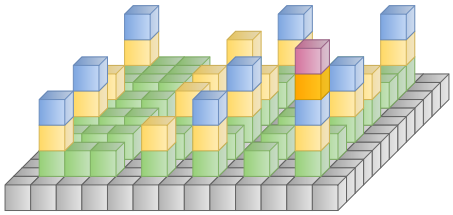
Our algorithm for the dynamic optimal transport problem employs two key strategies:

First, the dynamic optimal transport operations are reduced to simplex iterations. Our technique, grounded on the Simplex method, operates by eliminating the smallest

[1] Demo library is available at Github Repository: https://github.com/xyxu2033/DynamicOptimalTransport



(a) An example for 1D Euler Tour Tree



(b) Our 2D Euler Tour Tree

Figure 1: Overview of Euler Tour Tree with Skip Lists

cycle in the graph. We assume that the modifications influence only a small portion of the result, requiring only a few simplex iterations. It is worth noting that existing algorithms like the Network Simplex Algorithm perform poorly under dynamic modifications as they require scanning all the edges at least once to ensure the correctness of the solution.

Second, an efficient data structure is proposed for performing each simplex iteration within expected linear time complexity. Our data structure, as shown in Figure 1b, 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 the information about the *adjusted cost matrix*, which is a matrix that requires specific range modifications and queries.

The rest of the paper is organized as follows. In the "Preliminaries" section, we introduce several important definitions and notations that are used throughout this paper. In the "Skip Orthogonal List" section, we present the data structure *Skip Orthogonal List*, where "The Overall Structure" subsection explains how the data structure is organized and "The Update Operation: Cut" subsection uses *cut* operation as an example to demonstrate the updates on this data structure. In the "Dynamic Network Simplex Method" section we elaborate on how to use our data structure to solve the dynamic optimal transport problem, where the "Dynamic Optimal Transport" subsection shows that the dynamic optimal transport model can be reduced to simplex iterations, and the "Details for Simplex Iteration" subsection shows how our data structure could be used to improve the performance of each simplex iteration.

Preliminaries

Optimal Transport

Let A and B represent the supply and demand point sets respectively; the corresponding discrete probability distributions are $\alpha \in \mathbb{R}^A_{\geq 0}$ and $\beta \in \mathbb{R}^B_{\geq 0}$, such that $\sum_{a\in A} \alpha_a =$

$\sum_{b\in B}\beta_b = 1$. The cost matrix is $C \in \mathbb{R}^{A\times B}$ with each entry c_{ab} denoting the cost of transporting a unit of mass from the point $a \in A$ to the point $b \in B$. The discrete optimal transport problem can be formulated as (1).

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

Since Problem (1) is a standard network flow problem, it can be transformed to the following Problem (2) by adding infinity-cost edges (Peyré, Cuturi et al. 2019):

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

$$c_{a_1a_2} = c_{b_1a_1} = c_{b_1b_2} = +\infty \;\; \forall a_1, a_2 \in A, b_1, b_2 \in B \tag{3}$$

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

We add the constraint (3), and also redefine the point weights as (4). Note that in constraint (2), the input weight w must always satisfy $\sum_{v\in V} w_v = 0$; otherwise, the constraints cannot be satisfied.

We use X to denote a given basic solution in the context of using the simplex method to solve the Optimal Transport problem. We notice that the basic variables always form a spanning tree of the complete directed graph with self loops $G(V, V\times V)$ (Cunningham 1976). Let the dual variables be $\pi \in \mathbb{R}^V$, satisfying the following constraint:

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

We then define the adjusted cost matrix $C^\pi \in \mathbb{R}^{V\times V}$ as $c^\pi_{uv} \triangleq c_{uv} + \pi_v - \pi_u$, where c^π_{uv} represents the simplex multipliers for the linear program (Orlin 1997).

Euler Tour Technique

The *Euler Tour Technique* is a method for representing a tree T as a cyclic ordered set of linear length (Tarjan 1997). Specifically, given a tree $T(V, E_T)$, we construct a directed graph $D_T(V, E_D)$. The directed graph D_T shares the same vertex, and the edge set E_D is defined as follows:

- For each vertex $v \in V_T$, add the self-loop (v, v) to E_D;
- For each undirected edge $(u, v) \in E$, add two directed edges (u, v) and (v, u) to E_D.

Following this definition, $|E_D| = 2|E_T| + |V_T|$. Since T is a tree, $|E_T| = |V| - 1$, and therefore $|E_D| = 3|V| - 2 = O(|V|)$. Since the difference of In-Degree and Out-Degree of each vertex in D_T is 0, D_T should always contain an Euler Tour.

Definition 1 (Euler Tour Representation) *Given a tree T, the Euler Tour representation is an arbitrary sequence of Euler Tour of D_T represented by edges. That is, E_D with circular order induced by the Euler Tour is an Euler Tour representation.*

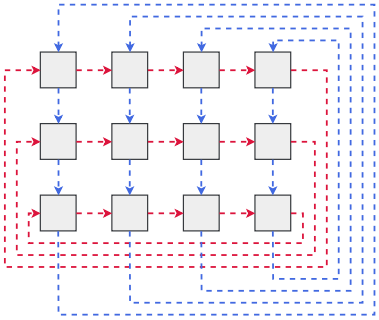


Figure 2: An Orthogonal List Example.

For the rest of the paper, E_D denotes the Euler Tour representation of the tree in the context.

Through Definition 1, we can reduce *edge linking*, *edge cutting*, *sub-tree weight updating* and *sub-tree weight querying* to constant number of *element insertion*, *element deletion*, *range weight updating* and *range weight querying* on a circular ordered set (Tarjan 1997). We show in the "Dynamic Network Simplex Method" section that the dynamic optimal transport can be reduced to the 2D version of these four operations.

Orthogonal Lists and Skip Lists

Skip Lists are the probabilistic data structures that extend a singly linked list with forward links at various levels, for improving the search, insertion, and deletion operations. Figure 1a illustrates an example for skip list. Each level contains a circular linked list, where the list at a higher level is a subset of the list at a lower level and the bottom level contains all the elements. The nodes at the same level have the same color and are linked horizontally. The corresponding elements in adjacent lists are connected by vertical pointers. We apply this skipping technique to circular singly linked lists in our work. Just as most self-balanced binary search trees, Skip Lists support "lazy propagation" techniques, allowing range modifications within $O(\log n)$ time, where n is the sequence length maintained by the tree (Sleator and Tarjan 1985). This technique is commonly used in dynamic trees for network problems (Tarjan 1997).

A k-dimensional *Orthogonal List* has k orthogonal forward links (it is a standard linked list when $k = 1$). Orthogonal lists, which can be singly, doubly, or circularly linked, can maintain the information mapped from the Cartesian product of ordered sets, such as sparse tensors (Butterfield, Ngondi, and Kerr 2016). Figure 2 demonstrates an orthogonal list that maintains a 3×4 matrix. Each node has 2 forward links, denoted by *row* links (red) and *column* links (blue). The *row* links connect the elements in each row into a circular linked list horizontally and the *column* links connect the elements in each column into a circular linked list vertically.

Skip Orthogonal List

In this section we introduce our novel data structure **Skip Orthogonal List**. In the "Dynamic Network Simplex

Method" section , this data structure is used for dynamically updating optimal transport. Formally, with the help of a Skip Orthogonal List, we can maintain a forest $F(V_F, E_F)$ with at most two trees, and a matrix $C^\pi \in \mathbb{R}^{V_F \times V_F}$ that supports the following operations. The first two and the last operations are for the case that F contains only one tree; the other two operations are for the case that F has two trees.

- **Cut**. Given an undirected edge (u, v), remove edge (u, v) from the tree, and split it into two disjoint trees. 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 F that does not connects with any other node. Let the original nodes form the vertex set V_1, and the new node itself form the vertex set V_2.
- **Range Update**. Given x, for each $(u, v) \in V \times V$, update c^π_{uv} as equation (6).

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

- **Link**. Given a pair $\{u, v\}$, add the edge (u, v) to the forest; connect two disjoint trees into a single tree, if u and v are disconnected.
- **Global Minimum Query**. Return the minimum value of C^π and its corresponding index.

For the remaining of the section, we construct a data structure with the expected $O(|V|^2)$ space complexity, where each operation can be done with the expected $O(|V|)$ time. "The Overall Structure" section shows the query process on this data structure and "The Update Operation: Cut" section illustrates the cut operation as an example based on this structure. For other operations (linking, insertion, and range updating), we leave them to our full version (Xu and Ding 2023) due to the space limit.

The Overall Structure

As shown in Figure 3, a **Skip Orthogonal List** is a hierarchical collection of Orthogonal lists, where 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.

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

- *$\{h(s)\}_{s \in S}$ is a set of $|S|$ independent random variables. The distribution is a geometric distribution with parameter p*
- *For each $\ell \in \mathbb{N}$, let L_ℓ be an Orthogonal List whose key contains all the elements in $S_\ell \times S_\ell$, where S_ℓ is the cyclic sequence formed by $S_\ell \triangleq \{s \in S \mid h_s \geq \ell\}$*

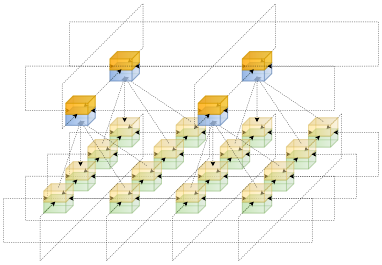


Figure 3: An illustration for Skip Orthogonal List.

Note that for any pair $u, v \in S$, we use (u, v) to denote the corresponding element in $S \times S$; with a slight abuse of notations, we also use "(u, v) at level ℓ to denote the corresponding node at the ℓ-th level in the Skip Orthogonal List.

We use this data structure to maintain several key information of $E_D \times E_D$. Since $|E_D| = O(|V|)$ as discussed in the "Euler Tour Technique" section, similar to conventional 1D Skip Lists, we know that the space complexity is $O(|V|^2)$ with high probability (due to the space limit, we place the proof in our full version).

Now we augment this data structure to store some additional information for range updating and global minimum query. Before that, the concept "dominate" needs to be adapted to 2D case defined as Definition 3.

Definition 3 *For any positive integer n, in a Skip Orthogonal List $\mathcal{L}$ over the cyclic ordered set S, suppose (u_1, v_1) and (u_2, v_2) are 2 elements in $S \times S$. We say the node (u_1, v_1) **dominates** (u_2, v_2) at level ℓ if and only if the following three conditions are all satisfied:*

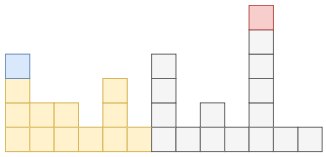
- *$h(u_1) \geq l$ and $h(v_1) \geq \ell$;*
- *$u_1 = u_2$ or $\max_{u=u_1+1}^{u_2} h(u) < \ell$;*
- *$v_1 = v_2$ or $\max_{v=v_1+1}^{v_2} h(v) < \ell$.*

Here, for any element s in the cyclic ordered set S, we use "$s + 1$" to denote the successor of s induced by the cyclic order.

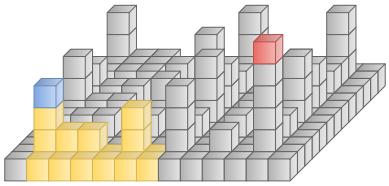
To better understand Definition 3, we illustrate the examples in Figure 4a and Figure 4b. In each figure, each blue node dominates itself and all the yellow nodes, while the red nodes dominate every node in the orthogonal list.

We now augment the Skip Orthogonal List of Definition 2. For each node (u, v) at orthogonal list L_ℓ, beside the two forward links and two backward links, we add the following attributes:

- **tag**: maintains the tag for lazy propagation for all the nodes dominated by it;
- **min_value**: maintains the minimum value among all the nodes dominated by it. Note that when $\ell = 0$, it stores the original value of c^π_{uv} following lazy propagation technique. That is, for any node x in the data structure, after each modification and query, the data structure needs to



(a) Domination in conventional 1D Skip List



(b) Domination in 2D Skip List

Figure 4: Illustrations of *Domination* in Skip Lists

assure

$$x.min\_value + \sum_{y \text{ dominates } x} y.tag = \min_{(u,v)\in S\times S, x \text{ dominates } (u,v)} c^{\pi}_{uv}.$$

- **min_index**: maintains the index corresponding to *min_value* attribute.
- **child**: points to (u, v) at the orthogonal list $L_{\ell-1}$ if $\ell > 0$, and it is invalid if $\ell = 0$;
- **parent**: points to the node that dominates it if $L_{\ell+1}$ is not empty.

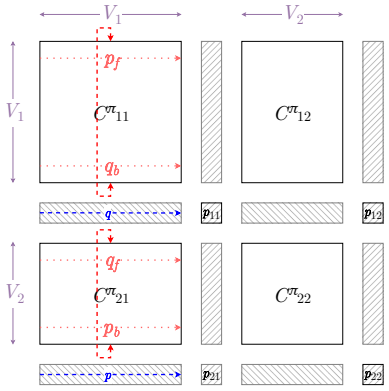
The Update Operation: Cut

In this subsection, we focus on the update operation "Cut" for a 2D Skip Orthogonal List as an example. Figure 5a and Figure 5b illustrate the basic idea of the cutting process.

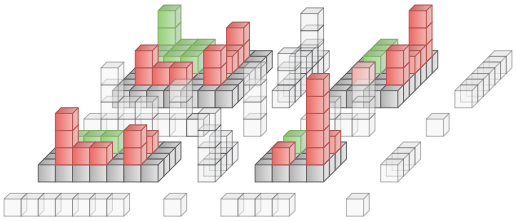
Taking an undirected edge (i, j) that needs to be cut as the input, the algorithm can be crudely described as follows:

1. Find the two rows and two columns representing the directed edges (i, j) and (j, i) in E_D, i.e. the transparent nodes in Figure 5b and the shaded nodes in Figure 5a.
2. Push down the *tag* attribute of all the nodes alongside the rows and columns, i.e. the red nodes and transparent nodes in Figure 5b. A node x needs to be pushed down, if some changes happen to the nodes dominated by x.
3. Cut the rows and columns, warping up the forward links and backward links of points alongside, as illustrated in Figure 5a. This operation cuts the original Skip Orthogonal List into four smaller lists.
4. Update the *min* attribute of the remaining nodes whose *tag* attribute was pushed down in step 2, i.e. the red nodes in Figure 5b.
5. Return the four smaller lists that were cut out in step 3.

The generalized lazy propagation to our 2D data structure ensures that only the nodes that are "close" to the two rows and columns are modified, and consequently the updating time is guaranteed to be low. Specifically, the expected time complexity is $O(|V|)$ upon each copy of procedure CUT.



(a) Vertical View (some notations in the figure are defined in our full version)



(b) 3D View

Figure 5: Illustrations of a "cut" operation

Dynamic Network Simplex Method

The simplex method performs simplex iterations on some initial feasible basis until the optimal solution is obtained. The simplex iterations are used for refining the current solution under the dynamic changes. In each simplex iteration, some variable with negative simplex multiplier is selected for a copy of procedure PIVOT, where one common strategy is to pivot in the variable with the smallest simplex multiplier. In the "Dynamic Optimal Transport Operations" section , we focus on defining the dynamic optimal transport operations and using simplex iterations to solve this problem, while in the "Details for Simplex Iteration" section , we analyze the details in each simplex iteration. Our method is presented in the context of the conventional Network Simplex algorithm (Cunningham 1976; Orlin 1997).

Dynamic Optimal Transport Operations

In an Optimal Transport problem, suppose the nodes in node set V are located in some metric space $\mathcal{X}$, e.g., the Euclidean Space $\mathbb{R}^k$. The edge cost c_{uv} is usually defined as the (squared) distance between u and v in the space. Let $w \in \mathbb{R}^V$ denote the weight vector as defined in the equation (4). A **Dynamic Optimal Transport** algorithm should support the following four types of update as well as online query:

- **Spatial Position Modification.** Select some supply or demand point $v \in V$ and move v to another point $v' \in \mathcal{X}$. This update usually results in the modification on 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

points $u, v \in V$ with some positive number $\delta \in \mathbb{R}_+$. Then update $w_u \leftarrow w_u - \delta$ and $w_v \leftarrow w_v + \delta$.

- **Point Deletion.** Delete a point $v \in V$ with $w_v = 0$ (before performing deletion, its weight should be already modified to be 0 via the above "weight modification", due to the requirement of weight balance for OT).
- **Point Insertion.** Select a point $v \notin V$; let $w_v = 0$ and insert v into set V (after the insertion, we can modify its weight from 0 to a specified value via the "weight modification").
- **Query.** Answer the current Optimal Transport plan and the cost.

These updates do not change the overall weights in the supply and demand sets, and thus $\sum_{v \in V} w_v \equiv 0$ and a feasible transport plan always exists. Therefore we can reduce these updates to the operations on simplex basis, and we explain the ideas below:

- **Spatial Position Modification.** The original optimal solution is primal feasible but not primal optimal, i.e. not dual feasible. We perform the primal simplex method based on the original optimal solution. When moving a point v, we first update the cost matrix C, the dual variable π and the modified cost C^π to meet the constraint (5). After that, we repeatedly perform the simplex iterations as long as the minimum value of the adjusted cost C^π is negative.
- **Weight Modification.** The original optimal solution is dual feasible but not primal feasible. We perform the dual simplex iterations based on the original optimal solution. Suppose we attempt to decrease w_u and increase w_v by δ. To implement this, we send δ amount of flow from u to v in the residual network by the similar manner of the shortest path augmenting method (Edmonds and Karp 1972). Specifically, we send the flow through basic variables. If some variable needs to be pivoted out before the required amount of flow is sent, we pivot in the variable with the smallest adjusted cost, and repeat this process.
- **Point Deletion & Point Insertion.** As the deleted/inserted point has weight 0 (even if the weight is non-zero, we can first perform the "weight modification" to modify it to be zero), whether inserting or deleting the point does not influence our result. We maintain a node pool keeping all the supply and demand nodes with 0 weight. Each **Point Insertion** operation takes some point from this pool and move it to the correct spatial location (i.e., insert a new point), while each **Point Deletion** operation returns a node to the pool.

Our solution updates the optimal transport plan as soon as an update happens, so we can answer the query for the optimal transport plan and value online. If the number of modified nodes is not large, intuitively the optimal transport plan should not change much, and thus we only need to run a small number of simplex iterations to obtain the OT solution. Assume we need to run s simplex iterations, where we assume $s \ll |V|$. Then the time complexity of our algorithm is $O(s \cdot \mathrm{Time}_{\mathrm{iter}})$ with $\mathrm{Time}_{\mathrm{iter}}$ being the time of each simplex iteration.

Details for Simplex Iteration

As discussed in the "Dynamic Optimal Transport Operations" section the dynamic operations on OT can be effectively reduced to simplex iterations. In this section, we review the operations used in the conventional network simplex algorithm, and show how to use the data structure designed in the "Skip Orthogonal List" section for maintaining C^π. The conventional network simplex method relies on the simplex method simplified by some graph properties. A (network) simplex iteration contains the following steps:

1. **Select Variable to Pivot in.** Select the variables with the smallest adjusted cost C^π to pivot in. Denote by $x_{i_{\mathrm{in}} j_{\mathrm{in}}}$ the selected one to be pivoted in.
2. **Update Primal Solution.** Adding the new variable to the current basis forms a cycle. We send the circular flow in the cycle, until some basic variable in the reverse direction runs out of flow, through Graph Search (e.g. Depth First Search) or Link/Cut Tree (Sleator and Tarjan 1981). Denote that node as $x_{i_{\mathrm{out}} j_{\mathrm{out}}}$, which is to be pivoted out.
3. **Update Dual Solution.** Update the dual variables π and modified cost C^π to meet the constraint (5), as the new basis, because C^π will soon be queried in the next simplex iteration.

The selecting step performs a query on the data structure on C^π for the minimum element, and the dual updating performs an update on the data structure. Though the primal updating step can be done within time $O(\log |V|)$ (Tarjan 1997), the conventional network simplex maintains C^π through brute force. That is, the conventional network simplex brutally traverses through all the adjusted costs and selects the minimum, and updates the adjusted cost one by one after the dual solution is updated. This indicates that the time complexity of each simplex iteration is $O(|V|^2)$. Our goal is to reduce this complexity; in particular, we aim to maintain C^π so that it can answer the global minimum query and perform update when the primal basis changes. In the simplex method, when we decide to pivot in the variable $x_{i_{\mathrm{in}} j_{\mathrm{in}}}$, we update the dual variables as the following equation (7),

$$\pi'_u \leftarrow \pi_u + \begin{cases} c^\pi_{i_{\mathrm{in}} j_{\mathrm{in}}} & u \in V_1 \\ 0 & u \in V_2 \end{cases} \tag{7}$$

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 $x_{i_{\mathrm{out}} j_{\mathrm{out}}}$ is cut; π and π' are respectively the dual solution before and after pivoting, where π_u and π'_u are the entries corresponding to the node u, for each $u \in V$.

Based on the definition of the adjusted cost matrix C^π, the update objective can be formulated as below:

$$c^{\pi'}_{uv} = c^\pi_{uv} + \pi'_u - \pi'_v = c^\pi_{uv} + \begin{cases} 0 & (u,v) \in V_1 \times V_1 \\ -c^\pi_{i_{\mathrm{in}} j_{\mathrm{in}}} & (u,v) \in V_1 \times V_2 \\ c^\pi_{i_{\mathrm{in}} j_{\mathrm{in}}} & (u,v) \in V_2 \times V_1 \\ 0 & (u,v) \in V_2 \times V_2 \end{cases}$$

where C^π is the adjusted cost matrix with regard to the old dual variables π while $C^{\pi'}$ regards the new dual variables π'. We present the details for updating the adjusted cost matrix

Algorithm 1: Adjusted Cost Matrix Update

Input: Adjusted cost matrix C^{π} corresponding to old solution π, entering variable $x_{i_{\text{in}}j_{\text{in}}}$, leaving variable $x_{i_{\text{out}}j_{\text{out}}}$.

Output: Updated adjusted cost matrix $C^{\pi'}$ corresponding to new dual solution π'.

1: $t \leftarrow c^{\pi}_{i_{\text{in}}j_{\text{in}}}$.
2: Cut the edge $(i_{\text{out}}, j_{\text{out}})$ in C^{π} and denote the 4 pieces as $C^{\pi}_{11}, C^{\pi}_{12}, C^{\pi}_{21}, C^{\pi}_{22}$.
3: Range update on C^{π}_{12} with increasing all entries by $-t$.
4: Range update on C^{π}_{21} with increasing all entries by t.
5: Link the pieces $\{C^{\pi}_{11}, C^{\pi}_{12}, C^{\pi}_{21}, C^{\pi}_{22}\}$ by the edge $x_{i_{\text{in}}j_{\text{in}}}$, and obtain the adjusted $C^{\pi'}$ as the output.

in Algorithm 1. Our Skip Orthogonal List presented in the "Skip Orthogonal List" section is capable of performing the operations *cut*, *add* and *link* in $O(|V|)$ time. Therefore we have the following Theorem 1.

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

Experiments

All the experimental results are obtained on a server equipped with *512GB* main memory of frequency *3200 MHz*; the data structures are implemented in *C++20* and compiled by *G++ 13.1.0* on *Ubuntu 22.04.3*. The data structures are compiled to shared objects by *PyBind11* (Jakob, Rhinelander, and Moldovan 2017) to be called by *Python 3.11.5*. Our code uses the Network Simplex library from (Bonneel et al. 2011) to obtain an initial feasible flow.

In our experiment, we use the **Network Simplex** algorithm (Orlin 1997) and **Sinkhorn** algorithm (Cuturi 2013) from the Python Optimal Transport (POT) library (Flamary et al. 2021). We test our algorithm for both the **Spatial Position Modification** and **Point Insertion** scenarios as described in the "Dynamic Optimal Transport Operations" section. Due to space limit, we place the experiments on **Weight Modification** and **Point Deletion** in our full version. We take execution time to measure their performances.

Datasets. We study the performance of our algorithm on both synthetic and real datasets. For synthetic datasets, we construct a mixture of two Gaussian distributions in $\mathbb{R}^{784}$, where the points of the same Gaussian distribution share the same label. We also use the popular real-world dataset MNIST (LeCun, Cortes, and Burges 2010). We partition the labels into two groups and compute the optimal transport between them.

Setup. We set the Sinkhorn regularization parameter λ as 0.1 and scale the median of the cost matrix C to be 1. We vary the the node size $|V|$ up to 4×10^4. For each dataset, we test the static execution time of POT on our machine and executed each dynamic operations 100 times to calculate the means of our algorithm. For *Spatial 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 outside the current OT instance to insert. We perform a query after these updates and compare with the static algorithms implemented in POT.

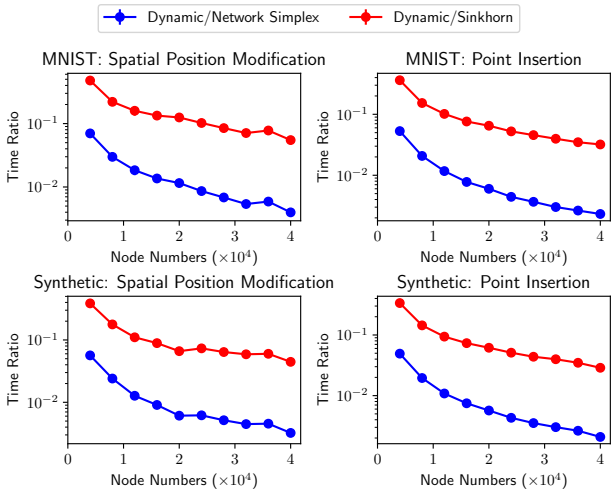


Figure 6: The ratio of the execution time of our dynamic algorithm to that of the static algorithms.

Result and Analysis. We illustrate our experimental results in Figure 6. In the dynamic scenarios, our algorithm is about 1000 times faster than static Network Simplex algorithm and 10 times faster than Sinkhorn Algorithm when the size $|V|$ reaches 40000, and reveals stable performance in practice. As the number of nodes grow larger, the advantage of our algorithm becomes more significant. This indicates that our algorithm is fast in the case when the number of simplex iterations is not large, as discussed in the "Dynamic Optimal Transport Operations" section. Also, the execution time of our algorithm is slightly higher than linear trend. Though each simplex iteration in our method is strictly linear in expectation theoretically, this could be influenced by several practical factors, such as the increment in number of simplex iterations, or the decrement in cache hit rate as the node size grows larger.

Conclusion and Future Work

In this paper, we propose a dynamic data structure for the traditional network simplex method. With the help of our data structure, the time complexity of the whole pivoting process is $O(|V|)$ in expectation. However, our algorithm lead to several performance issues in practice. First, as our algorithm stores the entire 2D Skip Orthogonal List data structure, it may consume relatively high space in practice. Second, as our algorithm is based on linked data structures, the cache hit rate is not high. An interesting future work for improving our implementation is to develop new algorithms and data structures with similar complexity but being more memory friendly.

Acknowledgements

The research of this work was supported in part by National Key R&D program of China through grant 2021YFA1000900, the NSFC throught Grant 62272432, and the Provincial NSF of Anhui through grant 2208085MF163. We also want to thank the anonymous reviewers for their helpful comments.

References

Alaya, M. Z.; Berar, M.; Gasso, G.; and Rakotomamonjy, A. 2019. Screening sinkhorn algorithm for regularized optimal transport. *Advances in Neural Information Processing Systems*, 32.

Alvarez-Melis, D.; and Fusi, N. 2020. Geometric Dataset Distances via Optimal Transport. In *NeurIPS 2020*. ACM.

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

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

Butterfield, A.; Ngondi, G. E.; and Kerr, A. 2016. *A dictionary of computer science*. Oxford University Press.

Chen, L.; Kyng, R.; Liu, Y. P.; Peng, R.; Gutenberg, M. P.; and Sachdeva, S. 2022. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, 612–623. IEEE.

Cheng, K.; Aeron, S.; Hughes, M. C.; and Miller, E. L. 2021. Dynamical Wasserstein barycenters for time-series modeling. *Advances in Neural Information Processing Systems*, 34: 27991–28003.

Cunningham, W. H. 1976. A network simplex method. *Mathematical Programming*, 11: 105–116.

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

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

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

Eppstein, D.; Goodrich, M. T.; and Sun, J. Z. 2005. The skip quadtree: a simple dynamic data structure for multidimensional data. In *Proceedings of the twenty-first annual symposium on Computational geometry*, 296–305.

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

Gramfort, A.; Peyré, G.; and Cuturi, M. 2015. 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*, 261–272. Springer.

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

Jakob, W.; Rhinelander, J.; and Moldovan, D. 2017. pybind11 – Seamless operability between C++11 and Python. ttps://github.com/pybind/pybind11. Accessed: 2023-05-11.

Janati, H.; Bazeille, T.; Thirion, B.; Cuturi, M.; and Gramfort, A. 2019. 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*, 743–754. Springer.

LeCun, Y.; Cortes, C.; and Burges, C. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist. Accessed: 2022-07-29.

Métivier, L.; Brossier, R.; Mérigot, Q.; Oudet, E.; and Virieux, J. 2016. 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.

Nickerson, B. G. 1994. Skip list data structures for multidimensional data. Technical report, University of Maryland at College Park, USA.

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

Pele, O.; and Werman, M. 2009. Fast and robust earth mover's distances. In *2009 IEEE 12th international conference on computer vision*, 460–467. IEEE.

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

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

Sherman, J. 2017. Generalized preconditioning and undirected minimum-cost flow. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, 772–780. SIAM.

Sleator, D. D.; and Tarjan, R. E. 1981. A data structure for dynamic trees. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, 114–122.

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

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

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

Thibault, A.; Chizat, L.; Dossal, C.; and Papadakis, N. 2021. Overrelaxed Sinkhorn–Knopp algorithm for regularized optimal transport. *Algorithms*, 14(5): 143.

Torres, L. C.; Pereira, L. M.; and Amini, M. H. 2021. A survey on optimal transport for machine learning: Theory and applications. *arXiv preprint arXiv:2106.01963*.

Van Den Brand, J.; Lee, Y. T.; Liu, Y. P.; Saranurak, T.; Sidford, A.; Song, Z.; and Wang, D. 2021. Minimum cost flows, mdps, and $\ell 1$-regression in nearly linear time for dense instances. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 859–869.

Xu, X.; and Ding, H. 2023. A Novel Skip Orthogonal List for Dynamic Optimal Transport Problem. *arXiv preprint arXiv:2310.18446*.