Fast Regularized Discrete Optimal Transport with Group-Sparse Regularizers

Yasutoshi Ida[1], Sekitoshi Kanai[1], Kazuki Adachi[1], Atsutoshi Kumagai[1], Yasuhiro Fujiwara[2]

[1]NTT Computer and Data Science Laboratories
[2]NTT Communication Science Laboratories
yasutoshi.ida@ieee.org, {sekitoshi.kanai.fu, kazuki.adachi.xy, atsutoshi.kumagai.ht, yasuhiro.fujiwara.kh}@hco.ntt.co.jp

Abstract

Regularized discrete optimal transport (OT) is a powerful tool to measure the distance between two discrete distributions that have been constructed from data samples on two different domains. While it has a wide range of applications in machine learning, in some cases the sampled data from only one of the domains will have class labels such as unsupervised domain adaptation. In this kind of problem setting, a group-sparse regularizer is frequently leveraged as a regularization term to handle class labels. In particular, it can preserve the label structure on the data samples by corresponding the data samples with the same class label to one group-sparse regularization term. As a result, we can measure the distance while utilizing label information by solving the regularized optimization problem with gradient-based algorithms. However, the gradient computation is expensive when the number of classes or data samples is large because the number of regularization terms and their respective sizes also turn out to be large. This paper proposes fast discrete OT with group-sparse regularizers. Our method is based on two ideas. The first is to safely skip the computations of the gradients that must be zero. The second is to efficiently extract the gradients that are expected to be nonzero. Our method is guaranteed to return the same value of the objective function as that of the original method. Experiments show that our method is up to 8.6 times faster than the original method without degrading accuracy.

Introduction

Regularized discrete optimal transport (OT) is a powerful tool to compute the distance between two discrete probability distributions that have been constructed from data samples on two different domains. It seeks a map, called a transportation plan, for moving the probability mass of one distribution to that of another distribution with the cheapest cost. Once the transportation plan is obtained, the data samples on one domain can be transported to those on another domain, and their distance can be computed as the cost based on the transportation plan. Owing to its theoretical foundations and desirable properties, it has received much attention in various fields, including shape recognition (Gangbo and McCann 2000), color transfer (Pitié, Kokaram, and Dahyot 2007), domain adaptation (Courty et al. 2017), and human activity recognition (Lu et al. 2021).

When we construct two distributions from data samples of two different domains on which we want to use regularized discrete OT, in some cases the sampled data from only one of the domains will have class labels, e.g., samples in the target domain are unlabeled, but labeled samples in similar domains can be observed. Such situations are frequently encountered in unsupervised/semi-supervised domain adaptation, which has received much attention in the machine-learning community (Courty et al. 2017). Although it is expected that utilizing not only data samples but also their class labels to obtain the transportation plan will improve the performance of such applications, the plain regularized discrete OTs, such as entropy regularized OT (Cuturi 2013), cannot handle class labels. To incorporate the information of the class labels into the transportation plan, a group-sparse regularizer is frequently used as a regularization term (Courty, Flamary, and Tuia 2014; Courty et al. 2017; Redko, Habrard, and Sebban 2017; Blondel, Seguy, and Rolet 2018; Das and Lee 2018a,b; Alaya et al. 2019; Li et al. 2020; Lu et al. 2021). In this approach, one group-sparse regularization term corresponds to one class label and the regularizer induces group-sparsity in the transportation plan. Specifically, this regularizer considers data samples with the same label as one group and restricts the transportation plan so that the samples tend to be transported to the same data sample on another domain (Figure 1). Since this approach preserves the structure of the label information on the data samples during the transportation, the group-sparse regularizer can effectively handle the problem of discrete OT with class labels.

While the discrete OT with a group-sparse regularizer is crucial for handling class labels, it incurs a high computation cost when solving the regularized optimization problem with a gradient-based algorithm, especially when there are large numbers of class labels (Russakovsky et al. 2015) or data samples in each class (Venturini, Baralis, and Garza 2017). This is because the gradient-based algorithm must compute gradient vectors corresponding to the regularization terms. Namely, when the numbers of class labels and number of data samples in each class increase, the number of gradient vectors and their sizes also increase. In addition, since the existing algorithms, such as L-BFGS (Blondel, Seguy, and Rolet 2018), need to iteratively compute these gradient vectors to update parameters until convergence, the compu-
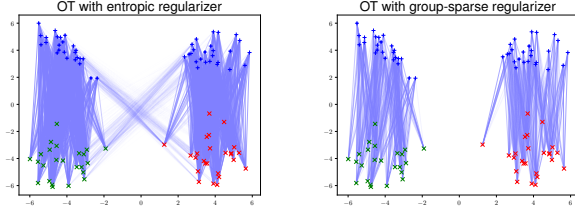
Figure 1: Map of transportation between samples on domain 1 with two classes (green and red samples) and those with two clusters on domain 2 (blue samples) for OT with entropic (left) and group-sparse (right) regularizers. In the left figure, a sample on domain 2 is transported from samples with different class labels, whereas it is transported from the same class in the right figure. These figures suggest that the group-sparse regularizer effectively preserves label information via the structured sparse transportation plan.

tation cost of the gradient computation becomes dominant in the total computation cost as the number of class labels or data samples in each class increases.

This paper proposes a fast algorithm for discrete OT with a group-sparse regularizer. To reduce the processing time, we utilize the observation that a large number of gradient vectors of the regularization terms turn out to be zero vectors during optimization. This is because the gradient vectors are computed with a soft-thresholding function such as the one used in Group Lasso (Yuan and Lin 2006; Ida, Fujiwara, and Kashima 2019; Ida et al. 2020), which induces sparsity in the gradient vectors. Based on the observation, our method leverages two ideas. The first idea is to safely skip gradient computations for groups whose gradient vectors must be zero vectors. A gradient vector for a group can be quickly checked to be a zero vector or not by approximately computing the soft-thresholding function. If it is determined to be a zero vector, our method skips the gradient computation; otherwise, it exactly computes the gradient vector. Here, when computing the exact gradient vector for the latter case, the checking procedure is an extra overhead. To alleviate this overhead, we introduce the second idea; we construct a subset of groups whose gradient vectors turn out to be nonzero vectors during optimization. We can reduce the overhead of the first idea by computing gradient vectors without the checking procedure for the specified set. Theoretically, our method is guaranteed to return the same value of the objective function as the original method. Experiments show that our method is up to 8.6 times faster than the original method without degrading accuracy.

Notation. We denote scalars, vectors, and matrices with lower-case, bold lower-case and upper-case letters, *e.g.*, t, $\boldsymbol{t}$ and T, respectively. Given a matrix T, we denote its elements by $t_{i,j}$ and its columns by $\boldsymbol{t}_j$. Given a vector $\boldsymbol{t}$, we denote its elements in the l-th group by $\boldsymbol{t}_{[l]}$ when $\boldsymbol{t}$ decomposes over groups $l \in \mathcal{L}$ where $\mathcal{L}$ is the set of group indices. $\boldsymbol{1}_m$ represents an m-dimensional vector whose elements are ones. We define $[\boldsymbol{x}]_+ := \max(\boldsymbol{x}, \boldsymbol{0})$ and $[\boldsymbol{x}]_- := \min(\boldsymbol{x}, \boldsymbol{0})$, performed element-wise.

Preliminary

First, we explain the transportation problem between two discrete probability distributions with label information. Next, we describe discrete OT with group-sparse regularizers that can handle label information. Finally, we introduce the smooth relaxed dual formulation of the discrete OT problem that is the key formulation to our method.

Transportation Problem with Label Information

We introduce two sets of data samples $\boldsymbol{X}_S = \{\boldsymbol{x}_S^{(i)} \in \mathbb{R}^d\}_{i=1}^m$ associated with class labels $\boldsymbol{Y}_S = \{y_S^{(i)} \in \mathcal{L}\}_{i=1}^m$ and unlabeled samples $\boldsymbol{X}_T = \{\boldsymbol{x}_T^{(i)} \in \mathbb{R}^d\}_{i=1}^n$. Note that $\mathcal{L} = \{1, ..., |\mathcal{L}|\}$ is the set of class labels.

Let us consider the transportation problem from $\boldsymbol{X}_S$ with $\boldsymbol{Y}_S$ to $\boldsymbol{X}_T$ by following the previous work (Courty et al. 2017). There, one first constructs two discrete probability distributions from the data samples $\boldsymbol{a} = \frac{1}{m} \sum_{i=1}^m \delta_{\boldsymbol{x}_S^{(i)}}$ and $\boldsymbol{b} = \frac{1}{n} \sum_{i=1}^n \delta_{\boldsymbol{x}_T^{(i)}}$ where $\delta_{\boldsymbol{x}^{(i)}}$ is the Dirac delta function at location $\boldsymbol{x}^{(i)}$. These distributions have masses of $1/m$ and $1/n$ for each $\delta_{\boldsymbol{x}_S^{(i)}}$ and $\delta_{\boldsymbol{x}_T^{(i)}}$, respectively.

Discrete OT seeks a map of the minimal cost in moving the mass of $\boldsymbol{a}$ to that of $\boldsymbol{b}$. As a result, we can transport $\boldsymbol{X}_S$ to $\boldsymbol{X}_T$ by using the map and compute the distance from the map as the cost. In Kantorovich's formulation, discrete OT is cast as a linear program (LP), as follows:

$$\min_{T \in \mathcal{U}(\boldsymbol{a}, \boldsymbol{b})} \langle T, C \rangle_F \tag{1}$$

where $\langle \cdot, \cdot \rangle_F$ is the Frobenius dot product and $\mathcal{U}(\boldsymbol{a}, \boldsymbol{b}) := \{T \in \mathbb{R}_+^{m \times n} : T\boldsymbol{1}_n = \boldsymbol{a}, T^\top \boldsymbol{1}_m = \boldsymbol{b}\}$ is the transportation polytope. T is a map called the transportation plan and $C \in \mathbb{R}_+^{m \times n}$ is a cost matrix. We set $c_{i,j} := \|\boldsymbol{x}_S^{(i)} - \boldsymbol{x}_T^{(j)}\|_2^2$ as in the previous work (Courty et al. 2017). The objective value after solving Problem (1) is the distance between the two discrete probability distributions. From here, we treat the samples as matrices, like $X^S \in \mathbb{R}^{m \times d}$, $\boldsymbol{y}^S \in \mathbb{R}^m$ and $X^T \in \mathbb{R}^{n \times d}$ that correspond to $\boldsymbol{X}_S$, $\boldsymbol{Y}_S$ and $\boldsymbol{X}_T$, respectively. In this case, when we obtain a transportation plan $\hat{T}$ by solving the LP problem, X^S is transported to X^T as $n\hat{T}^\top X^S$. However, this formulation cannot handle class labels $\boldsymbol{y}^S$, which may improve accuracy of some applications, such as unsupervised domain adaptation. The next section describes how to find a transportation plan via discrete OT while considering class labels.

Discrete OT with Group-sparse Regularizer

To handle class labels $\boldsymbol{y}^S$, a group-sparse regularizer $\Psi(\cdot)$ (Blondel, Seguy, and Rolet 2018) can be incorporated in Problem (1) as follows:

$$\min_{T \in \mathcal{U}(\boldsymbol{a}, \boldsymbol{b})} \langle T, C \rangle_F + \sum_{j=1}^n \Psi(\boldsymbol{t}_j), \tag{2}$$

where

$$\Psi(\boldsymbol{t}_j) = \gamma(\tfrac{1}{2}\|\boldsymbol{t}_j\|_2^2 + \mu \sum_{l \in \mathcal{L}} \|\boldsymbol{t}_{j[l]}\|_2). \tag{3}$$

$\gamma > 0$ and $\mu > 0$ are hyperparameters for the regularization terms. $\boldsymbol{t}_{j[l]}$ is a vector containing elements corresponding to

the labeled data associated with the class label $l \in \mathcal{L}$. The term $\|\boldsymbol{t}_{j[l]}\|_2$ can be expected to make $\boldsymbol{t}_{j[l]}$ a zero vector, as in Group Lasso (Yuan and Lin 2006). Therefore, we can expect the transportation plan T to be a sparse one reflecting the group structure, which corresponds to the structure of data samples with class labels. Specifically, this regularizer considers data samples X^S with the same label as one group and restricts the transportation plan T so that they tend to be transported to the same data sample of X^T (Figure 1). Although solving Problem (2) does not seem easy, the next section shows that it can easily be solved by considering the smooth relaxed dual of Problem (2).

Smooth Relaxed Dual Formulation

The smooth relaxed dual of Problem (2) is as follows (Blondel, Seguy, and Rolet 2018):

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^m, \boldsymbol{\beta} \in \mathbb{R}^n} \boldsymbol{\alpha}^\top \boldsymbol{a} + \boldsymbol{\beta}^\top \boldsymbol{b} - \sum_{j=1}^n \psi(\boldsymbol{\alpha} + \beta_j \mathbf{1}_m - \boldsymbol{c}_j). \quad (4)$$

In the above problem, $\psi(\boldsymbol{f}) := \sup_{\boldsymbol{g} \geq 0} \boldsymbol{f}^\top \boldsymbol{g} - \Psi(\boldsymbol{g})$ is the convex conjugate of $\Psi(\cdot)$. Specifically, it is computed as $\psi(\boldsymbol{f}) = \boldsymbol{f}^\top \boldsymbol{g}^\star - \Psi(\boldsymbol{g}^\star)$, where $\boldsymbol{g}^\star$ decomposes over groups $l \in \mathcal{L}$ and equals:

$$
\begin{aligned}
\boldsymbol{g}_{[l]}^\star &= \arg\min_{\boldsymbol{g}_{[l]}} \tfrac{1}{2}\|\boldsymbol{g}_{[l]} - \boldsymbol{f}_{[l]}^+\|_2^2 + \mu\|\boldsymbol{g}_{[l]}\|_2 \\
&= [1 - \mu/\|\boldsymbol{f}_{[l]}^+\|_2]_+ \boldsymbol{f}_{[l]}^+ = \nabla\psi(\boldsymbol{f})_{[l]}. \quad (5)
\end{aligned}
$$

In the above equations, $\boldsymbol{f}^+ = \frac{1}{\gamma}[\boldsymbol{f}]_+$ and the third equation is derived from the definition of $\psi(\cdot)$ and Danskin's theorem (Bertsekas 1999). The optimal transportation plan $T^\star$ of Problem (2) can be recovered from the optimal solutions $\boldsymbol{\alpha}^\star$ and $\boldsymbol{\beta}^\star$ by computing $\boldsymbol{t}_j^\star = \nabla\psi(\boldsymbol{\alpha}^\star + \beta_j^\star \mathbf{1}_m - \boldsymbol{c}_j)$ for all $j \in \{1, ..., n\}$. Problem (4) is a differentiable and concave optimization problem without hard constraints. In addition, we can compute the gradient $\nabla\psi(\boldsymbol{\alpha} + \beta_j \mathbf{1}_m - \boldsymbol{c}_j)$ in a closed-form expression by using Equation (5). Therefore, we can use gradient-based algorithms, such as L-BFGS (Liu and Nocedal 1989), to solve Problem (4).

Equation (5) can be regarded as a soft-thresholding function (Fujiwara et al. 2016a,b). Since the $[1 - \mu/\|\boldsymbol{f}_{[l]}^+\|_2]_+$ part in Equation (5) is represented as $\max(1 - \mu/\|\boldsymbol{f}_{[l]}^+\|_2, 0)$, a lot of gradient vectors $\nabla\psi(\boldsymbol{\alpha} + \beta_j \mathbf{1}_m - \boldsymbol{c}_j)_{[l]}$ turn out to be zero vectors during optimization. However, the gradient vector $\nabla\psi(\boldsymbol{\alpha} + \beta_j \mathbf{1}_m - \boldsymbol{c}_j)_{[l]}$ is computed for all $l \in \mathcal{L}$ in every iteration until convergence. If g is the number of data samples per class, $\mathcal{O}(|\mathcal{L}|ng)$ time is required for computing all the gradient vectors in one iteration. When the total computation cost other than the gradient computation of the solver is $\mathcal{O}(s_s)$ time, and the number of iterations until convergence is s_t, the algorithm requires $\mathcal{O}(|\mathcal{L}|ngs_t + s_s)$ time. This leads to a long processing time as the numbers of class labels $|\mathcal{L}|$, data samples in each class g, or unlabeled data samples n increases.

Proposed Approach

We first outlines our ideas to efficiently solve Problem (4). Next, we explain these ideas in detail and introduce our algorithm. The proofs can be found in the supplementary file.

Ideas

The algorithm takes a long time to solve Problem (4) when the numbers of class labels and data samples are large. This is because it requires $\mathcal{O}(|\mathcal{L}|ng)$ time to compute the gradient vectors $\nabla\psi(\cdot)$ for each iteration until convergence, as described in the previous section.

To accelerate the gradient computation, we introduce two ideas. The first idea is to skip the gradient computations for groups whose gradient vectors must be zero vectors. As shown in Equation (5), many of the gradient vectors during optimization are expected to be zero vectors owing to the soft-thresholding function. On the basis of this observation, we approximately compute the soft-thresholding function at $\mathcal{O}(|\mathcal{L}|(n + g))$ time and quickly check whether the gradient vectors must be zero vectors or not in advance of the exact gradient computation.

In the first idea, if a gradient vector turns out to be a nonzero vector, the checking procedure incurs unnecessary cost as it does not skip the computation of that gradient. The second idea is to identify a subset of nonzero gradient vectors and compute them in the specified set without the checking procedure of the first idea. As a result, it reduces the overhead of the first idea.

Skipping Gradient Computations

This section details the first idea of skipping the gradient computations for groups whose gradient vectors must be zero vectors. To identify such groups, we introduce a criterion that is computed as follows:

Definition 1 *Suppose that $z_{l,j} := \|[(\boldsymbol{\alpha} + \beta_j \mathbf{1}_m - \boldsymbol{c}_j)_{[l]}]_+\|_2$. Let $\tilde{z}_{l,j}$, $\tilde{\boldsymbol{\alpha}}$ and $\tilde{\boldsymbol{\beta}}$ be old versions (snapshots) of $z_{l,j}$, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ at some iteration in a gradient-based algorithm for optimization of Problem (4). We define $\overline{Z} \in \mathbb{R}_+^{|\mathcal{L}| \times n}$ and $\overline{z}_{l,j} \in \overline{Z}$ is computed as follows:*

$$\overline{z}_{l,j} = \tilde{z}_{l,j} + \|[\Delta\boldsymbol{\alpha}_{[l]}]_+\|_2 + \sqrt{g_l}[\Delta\beta_j]_+, \quad (6)$$

where $\Delta\boldsymbol{\alpha} = \boldsymbol{\alpha} - \tilde{\boldsymbol{\alpha}}$, $\Delta\boldsymbol{\beta} = \boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}$, and g_l represents the size of the l-th group.

The snapshots are taken at regular intervals in the gradient-based algorithm, e.g., every ten iterations, in the iterative method. Note that $z_{l,j}$ is a quantity that is used to compute the gradient vector $\nabla\psi(\boldsymbol{f})_{[l]}$ in Equation (5) because $\nabla\psi(\boldsymbol{f})_{[l]} = [1 - \mu/\|\boldsymbol{f}_{[l]}^+\|_2]_+ \boldsymbol{f}_{[l]}^+ = [1 - \mu\gamma/z_{l,j}]_+ \boldsymbol{f}_{[l]}^+$. The following lemma shows that $\overline{z}_{l,j}$ is an upper bound of $z_{l,j}$:

Lemma 1 (Upper Bound) $\overline{z}_{l,j} \geq z_{l,j}$ *holds when $\overline{z}_{l,j}$ is computed using Equation (6).*

From the above lemma, we have the following lemma:

Lemma 2 *When $\mu\gamma \geq \overline{z}_{l,j}$ holds, we have $\nabla\psi(\boldsymbol{\alpha} + \beta_j \mathbf{1}_m - \boldsymbol{c}_j)_{[l]} = \mathbf{0}$.*

The above lemma shows that we can identify groups whose gradient vectors must be zero vectors by utilizing the upper bound $\overline{z}_{l,j}$. The cost of computing $\overline{z}_{l,j}$ is as follows:

Lemma 3 *Given snapshots $\tilde{Z}$, $\tilde{\boldsymbol{\alpha}}$, and $\tilde{\boldsymbol{\beta}}$, the cost of computing Equation (6) for all elements in $\overline{Z} \in \mathbb{R}_+^{|\mathcal{L}| \times n}$ is $\mathcal{O}(|\mathcal{L}|(n + g))$ time.*

The above lemma suggests that leveraging the upper bound allows us to efficiently identify groups whose gradient vectors must be zero vectors. This is because the computation cost is $\mathcal{O}(|\mathcal{L}|(n+g))$ time, while the original method requires $\mathcal{O}(|\mathcal{L}|ng)$ time.

As described above, our upper bounds efficiently skip the gradient computations whose gradient vectors turn out to be zero vectors. However, when a gradient vector turns out to be a nonzero vector, we must compute the upper bound as well as the exact gradient vector. The next section shows a way to avoid this problem.

Reduction of Overhead

This section details our second idea: constructing a subset of groups whose gradient vectors turn out to be nonzero vectors. In the first idea described above, we compute the upper bound $\overline{z}_{l,j}$ by using Equation (6) and skip the gradient computation if $\mu\gamma \geq \overline{z}_{l,j}$ holds from Lemma 1. However, if $\mu\gamma \geq \overline{z}_{l,j}$ does not hold, we must compute the gradient vector by using Equation (5). In this case, we have to compute both Equations (6) and (5). This may incur a large overhead if many gradient computations cannot be skipped.

To reduce the overhead of computing the upper bound, our second idea identifies a subset of nonzero gradient vectors during optimization and computes gradient vectors in the specified subset without computing the upper bound. We introduce the following criterion to identify the subset by utilizing the same variables definitions in Definition 1:

Definition 2 *Suppose that* $\tilde{k}_{l,j} := \|(\tilde{\boldsymbol{\alpha}} + \tilde{\beta}_j \mathbf{1}_m - \boldsymbol{c}_j)_{[l]}\|_2$ *and* $\tilde{o}_{l,j} := \|[(\tilde{\boldsymbol{\alpha}} + \tilde{\beta}_j \mathbf{1}_m - \boldsymbol{c}_j)_{[l]}]_-\|_2$ *are snapshots similar to those in Definition 1. We define* $\underline{Z} \in \mathbb{R}^{|\mathcal{L}| \times n}$ *and* $\underline{z}_{l,j} \in \underline{Z}$ *is computed as follows:*

$$
\begin{aligned}
\underline{z}_{l,j} &= \tilde{k}_{l,j} - \|\Delta\boldsymbol{\alpha}_{[l]}\|_2 - \sqrt{g_l}\|\Delta\beta_j\|_2 \\
&\quad - \tilde{o}_{l,j} - \|[\Delta\boldsymbol{\alpha}_{[l]}]_-\|_2 - \sqrt{g_l}\|[\Delta\beta_j]_-\|_2. \quad (7)
\end{aligned}
$$

The following lemma shows that $\underline{z}_{l,j}$ is a lower bound of $z_{l,j}$ in Definition 1:

Lemma 4 (Lower Bound) $\underline{z}_{l,j} \leq z_{l,j}$ *holds when* $\underline{z}_{l,j}$ *is computed by Equation (7).*

From the above lemma, we have the following lemma:

Lemma 5 *When* $\mu\gamma < \underline{z}_{l,j}$ *holds, we obtain* $\nabla\psi(\boldsymbol{\alpha} + \beta_j \mathbf{1}_m - \boldsymbol{c}_j)_{[l]} \neq \mathbf{0}$.

According to this lemma, we can identify nonzero gradient vectors by leveraging the lower bound $\underline{z}_{l,j}$. Therefore, we can construct the subset of groups that have nonzero gradient vectors as follows:

Definition 3 *The set* $\mathbb{N}$ *is constructed as*

$$
\mathbb{N} = \{(l,j) \in \{1,...,|\mathcal{L}|\} \times \{1,...,n\} | \mu\gamma < \underline{z}_{l,j}\}. \quad (8)
$$

The computation cost is as follows:

Lemma 6 *Given snapshots* $\tilde{K}$, $\tilde{O}$, $\tilde{\boldsymbol{\alpha}}$, *and* $\tilde{\boldsymbol{\beta}}$, *the computation cost of constructing the set* $\mathbb{N}$ *is* $\mathcal{O}(|\mathcal{L}|(n+g))$ *time.*

Algorithm 1: Fast OT with Group Regularizer

1: $\boldsymbol{\alpha} \leftarrow \mathbf{0}, \boldsymbol{\beta} \leftarrow \mathbf{0}, \tilde{\boldsymbol{\alpha}} \leftarrow \mathbf{0}, \tilde{\boldsymbol{\beta}} \leftarrow \mathbf{0}, \mathbb{N} = \emptyset$
2: **repeat**
3:     apply a solver to Problem 4 with the function GRADPSI in Algorithm 2 for r iterations;
4:     compute $\|\Delta\boldsymbol{\alpha}_{[l]}\|_2$ and $\|[\Delta\boldsymbol{\alpha}_{[l]}]_-\|_2$ for $l \in \mathcal{L}$;
5:     compute $\Delta\boldsymbol{\beta}$ and $[\Delta\boldsymbol{\beta}]_-$;
6:     $\mathbb{N} = \emptyset$;
7:     **for** each $j \in \{1,...n\}$ **do**
8:         **for** each $l \in \mathcal{L}$ **do**
9:             compute the lower bound $\underline{z}_{l,j}$ by Eq. (7);
10:            **if** $\underline{z}_{l,j} > \mu\gamma$ **then**
11:                add (j,l) to $\mathbb{N}$;
12:            **end if**
13:        **end for**
14:    **end for**
15:    update the snapshots;
16: **until** $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ converge

---

Algorithm 2: Gradient Computation of $\nabla\psi(\boldsymbol{\alpha} + \beta_j\mathbf{1}_m - \boldsymbol{c})$

1: **function** GRADPSI:
2:     **for** $(l,j) \in \mathbb{N}$ **do**
3:         compute $\nabla\psi(\boldsymbol{\alpha}_{[l]} + \beta_j\mathbf{1}_{gl} - \boldsymbol{c}_{j[l]})$;
4:     **end for**
5:     compute $\|[\Delta\boldsymbol{\alpha}_{[l]}]_+\|_2$ for $l \in \mathcal{L}$ and $[\Delta\boldsymbol{\beta}]_+$;
6:     **for** $(l,j) \notin \mathbb{N}$ **do**
7:         compute the upper bound $\overline{z}_{l,j}$ by Eq. (6);
8:         **if** $\overline{z}_{l,j} \leq \mu\gamma$ **then**
9:             $\nabla\psi(\boldsymbol{\alpha}_{[l]} + \beta_j\mathbf{1}_{gl} - \boldsymbol{c}_{j[l]}) \leftarrow \mathbf{0}$;
10:        **else**
11:            compute $\nabla\psi(\boldsymbol{\alpha}_{[l]} + \beta_j\mathbf{1}_{gl} - \boldsymbol{c}_{j[l]})$;
12:        **end if**
13:    **end for**
14:    **return** $[\nabla\psi(\boldsymbol{\alpha}+\beta_1\mathbf{1}_m-\boldsymbol{c}),...,\nabla\psi(\boldsymbol{\alpha}+\beta_n\mathbf{1}_m-\boldsymbol{c})]$;
15: **end function**

The proof is similar to that of Lemma 3. After constructing the set $\mathbb{N}$, the gradient vectors corresponding to $\mathbb{N}$ are computed by using Equation (5) without the checking procedure of Lemma 2. As a result, our method can reduce the total overhead computing the upper bound since it does not compute the upper bound corresponding to $\mathbb{N}$. Note that although the cost of the lower bound is the same as that of the upper bound, the total cost of the lower bound is smaller than that of the upper bound. This is because the set $\mathbb{N}$ is constructed at regular intervals during optimization. The procedure is described in the next section.

Algorithm

Algorithm 1 is the pseudocode of our algorithm. Although it applies a solver, such as L-BFGS, to Problem (4), the solver efficiently computes $\nabla\psi(\boldsymbol{\alpha} + \beta_j\mathbf{1}_m - \boldsymbol{c})$ by utilizing the upper bounds, as we described in Lemma 2. Algorithm 2 is the gradient computation. Here, the gradient vectors corresponding to $\mathbb{N}$ are computed as usual by following Lemma 5 (lines 2–4). The other gradient vectors are computed with

the upper bounds (lines 6–13). Namely, we first compute the upper bound $\overline{z}_{l,j}$ on the basis of Equation (6) (line 7) and skip the gradient computation if $\overline{z}_{l,j} \leq \mu\gamma$ holds by following Lemma 2 (lines 8–9). If the inequality does not hold, the algorithm does not skip the computation (lines10–11).

In Algorithm 1, after it initializes the parameters, snapshots, and set $\mathbb{N}$ (line 1), it calls the solver with Algorithm 2 for r iterations (line 3). We set $r = 10$ in this paper. Next, it performs precomputations to obtain lower bounds for constructing the set $\mathbb{N}$ whose gradient vectors are expected to be nonzero vectors (lines 4–5). It uses Equation (7) to compute the lower bound $\underline{z}_{l,j}$ (line 9) and adds the index to $\mathbb{N}$ if $\underline{z}_{l,j} > \mu\gamma$ holds on the basis of Lemma 5 (lines 10–12). Then, it updates the snapshots (line 15). It repeats the above procedure until convergence (line 16).

The computation cost of Algorithm 1 is as follows:

Theorem 1 (Computation Cost) *In Algorithm 1, let s_i and s_n be the total number of $(j, l) \in \mathbb{N}$ and $(j, l) \notin \mathbb{N}$ for all iterations, respectively. Suppose that s_u is the total number of un-skipped gradient computations on line 11, and s_r is the total number of loops of lines 2–16. If $\mathcal{O}(s_s)$ time is the total computation cost other than the gradient computation of the solver in Algorithm 1, it requires $\mathcal{O}((|\mathcal{L}|ns_r + s_i + s_u)g + s_n + s_s)$ time.*

If many gradient vectors become zero vectors during optimization, s_i and s_u are expected to be small. Since the original method requires $\mathcal{O}(|\mathcal{L}|ngrs_r + s_s)$ time from $s_t = rs_r$, our method can be faster than the original method when the gradients are sparse. Note that since our method reduces the cost of the gradient computation, it can be used with a wide range of solvers, such as L-BFGS. Therefore, the total computation cost changes depending on the cost of the solver, i.e., s_s in the total computation cost in Theorem 1.

In terms of the optimization result, Algorithm 1 has the following property:

Theorem 2 (Optimization Result) *Suppose that Algorithm 1 has the same hyperparameters as those of the original method. Then, Algorithm 1 converges to the same solution and objective value as those of the original method.*

The above theorem clearly holds because our method exactly computes all the nonzero gradient vectors in lines 2–4 and 6–13 in Algorithm 2. It suggests that our algorithm efficiently solves Problem (4) without degrading accuracy.

Convergence of Bounds

Although Theorem 2 guarantees the optimization results of the solution and the objective value after convergence, our bounds also have advantageous properties for convergence. Specifically, the upper bound has the following property:

Theorem 3 (Convergence of Upper Bound) *Let $\overline{\epsilon}$ be an error bound defined as $|\overline{z}_{l,j} - z_{l,j}|$. Then, we have $\overline{\epsilon} = 0$ when $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ reaches convergence through a gradient-based algorithm.*

The above theorem suggests that the upper bound $\overline{z}_{l,j}$ of $z_{l,j}$ converges to the exact value of $z_{l,j}$ if the algorithm converges. This indicates that if $\mu\gamma \geq z_{l,j}$ holds after convergence, $\mu\gamma \geq \overline{z}_{l,j}$ always holds in Lemma 2. When $\mu\gamma \geq z_{l,j}$

holds, we have $\nabla\psi(\boldsymbol{\alpha}+\beta_j\mathbf{1}_m-\boldsymbol{c}_j)_{[l]} = \mathbf{0}$ from the proof of Lemma 2. Therefore, our upper bound can exactly identify all the gradient vectors that turn out to be zero vectors when the algorithm converges.

As for the lower bound, we have the following property:

Theorem 4 (Convergence of Lower Bound) *Let $\underline{\epsilon}$ be an error bound defined as $|z_{l,j} - \underline{z}_{l,j}|$. Suppose that $\boldsymbol{f} := \boldsymbol{\alpha} + \beta_j\mathbf{1}_m - \boldsymbol{c}_j$. If $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ reaches convergence through a gradient-based algorithm, we have $\underline{\epsilon} = |\|[\boldsymbol{f}_{[l]}]_+\|_2 + \|[\boldsymbol{f}_{[l]}]_-\|_2 - \|\boldsymbol{f}_{[l]}\|_2|$.*

Although the above theorem suggests that the error bound of the lower bound does not converge to zero, this error bound has the following advantage:

Corollary 1 (Convergence of Lower Bound) *If we have $[\boldsymbol{f}_{[l]}]_+ = \mathbf{0}$ or $[\boldsymbol{f}_{[l]}]_- = \mathbf{0}$ for Theorem 4, we have $\underline{\epsilon} = 0$.*

The above corollary indicates that the lower bound is tight when all the elements in $\boldsymbol{f}_{[l]}$ are positive or negative. This suggests that when a lot of gradient vectors turn out to be zero vectors during optimization, the lower bound is expected to be tight since $[\boldsymbol{f}_{[l]}]_+ = \mathbf{0}$ will hold in many cases. For nonzero gradient vectors, we can obtain a tight bound if all the elements in the gradient vector are nonzero because $[\boldsymbol{f}_{[l]}]_- = \mathbf{0}$ holds in Corollary 1.

Related Work

To handle the group structure of the transportation plan, Courty, Flamary, and Tuia (2014) utilized entropic and $\ell_p - \ell_1$ regularization terms with $p < 1$. Although the regularization term made the objective function nonconvex, they solved the optimization problem by using a majoration-minimization algorithm. However, the method is only guaranteed to converge to local stationary points. This drawback can be overcome by using a $\ell_1 - \ell_2$ regularization term instead of the $\ell_p - \ell_1$ regularization term (Courty et al. 2017). This approach outperformed the previous method (Courty, Flamary, and Tuia 2014) on many benchmark tasks (Courty et al. 2017) and its regularization term is now widely used to enable transportation plans to handle group structure (Redko, Habrard, and Sebban 2017; Das and Lee 2018a,b; Li et al. 2020; Lu et al. 2021). However, it does not achieve group sparsity, as pointed out in the previous work (Blondel, Seguy, and Rolet 2018). This is because the logarithm in the entropic regularization term keeps the values of the transportation plan in the strictly positive orthant. Therefore, it is difficult to obtain a sparse transportation plan by using this approach. On the other hand, Blondel, Seguy, and Rolet (2018) proposed another group-sparse regularizer, as we described in relation to Equation (3). Since this regularizer leverages the squared 2-norm and $\ell_1 - \ell_2$ regularization terms, it can avoid the above limitation. Namely, their regularization term truly achieves group sparsity owing to the soft-thresholding function of Equation (5). In addition, the gradients can be computed in a closed-expression, and we can use various solvers on the optimization problem, as we explained in the preliminary section. However, the cost of computing the gradient tends to be high when we handle large datasets because it requires $\mathcal{O}(|\mathcal{L}|ng)$ time to compute the gradient vectors for each iteration until convergence.
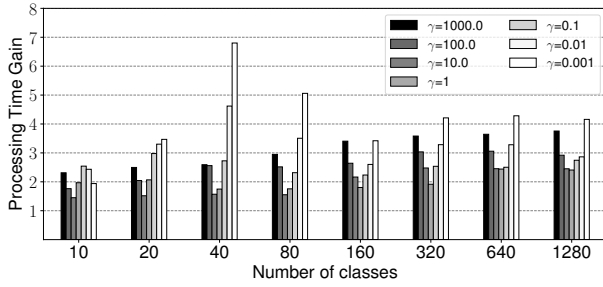
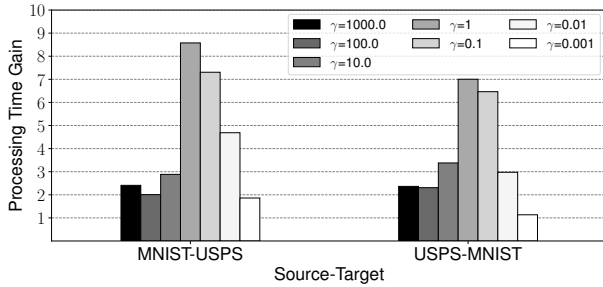Figure 2: Processing time gain for each hyper parameter when the numbers of classes change.

Figure 4: Processing time gain of 12 adaptation tasks on face recognition.
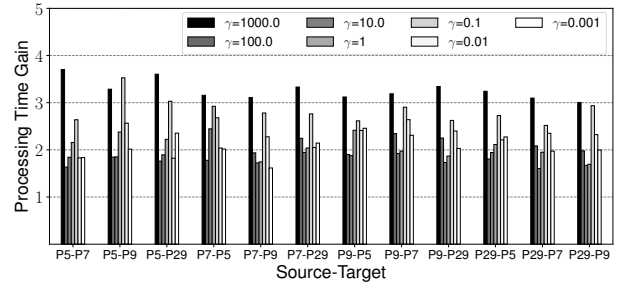

Figure 3: Processing time gain of 2 adaptation tasks on digit recognition.
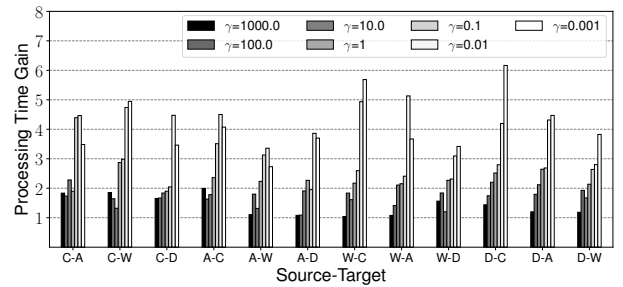
Figure 5: Processing time gain of 12 adaptation tasks on object recognition.

Experiment

We evaluated the processing time and accuracy to confirm the efficiency and effectiveness of our method.

Datasets

We created X^S, X^T, and $\boldsymbol{y}^S$ from the following datasets including a synthetic dataset and visual adaptation datasets in accordance with the previous work (Courty et al. 2017):

Synthetic dataset with controlled number of class labels. We used a simulated dataset with controlled numbers of class labels and data samples to show the efficiency. We increased the number of class labels $|\mathcal{L}|$ from 10 to 1,280 for X^S. The number of dimensions for each data sample was two. Each class had ten data samples ($g = 10$), which were generated from a standard normal distribution with a different mean for each class. The means were computed as $(l \times 5.0, -5.0)$ for X^S and $(l \times 5.0, 5.0)$ for X^T where $l \in \mathcal{L}$. The labels l of X^T were only used to generate X^T and not used for the optimization. Note that the numbers of data samples m and n automatically increased from 100 to 12,800 because we set $n = m$ and $m = |\mathcal{L}|g$.

Digit recognition. We used the digits datasets: USPS (U) (Hull 1994) and MNIST (M) (Lecun et al. 1998) as X^S and X^T. Both datasets have ten class labels of digits. We randomly sampled 5,000 images from each dataset. The images in the datasets were resized to 16×16.

Face recognition. We used the PIE dataset for the face

recognition task (Gross et al. 2008). It contains 32×32 images of 68 individuals taken under various conditions. The number of classes is 68. We used four domains in the dataset: PIE05 (P5), PIE07 (P7), PIE09 (P9), and PIE29 (P29). We created the combination of X^S and X^T by choosing two domains from these four domains. As a result, we had 12 transportation problems for this dataset. The numbers of images are 3332 (P5), 1629 (P7), 1632 (P9), and 1632 (P29).

Object recognition. We used the Caltech-Office dataset for the object recognition task with ten class labels (Griffin, Holub, and Perona 2007; Gong et al. 2012). The dataset consisted of four domains: *Caltech-256* (C), *Amazon* (A), *Webcam* (W) and *DSLR* (D). Therefore, we had 12 transportation problems for this dataset. The samples numbered 1123, 958, 295 and 157, respectively. We used DeCAF$_6$ (Donahue et al. 2014) as the feature vectors. They are activations of the fully connected layer of a convolutional neural network trained on ILSVRC-12. The size of the vectors was 4096.

Experimental Setup

We evaluated the processing time of solving Problem (4) between different domains. By following the implementation of the previous work (Blondel, Seguy, and Rolet 2018), we used L-BFGS and the hyperparameter $\rho \in [0, 1]$ instead of μ in Equation (3) to balance the regularization terms. Namely, we utilized $\Psi(\boldsymbol{t}_j) = \gamma(\frac{1}{2}(1 - \rho)\|\boldsymbol{t}_j\|_2^2 + \rho \sum_{l \in \mathcal{L}} \|\boldsymbol{t}_{j[l]}\|_2)$ instead of Equation (3). We eval-

uated the processing time and accuracy on combinational settings of the hyperparameters $\rho = \{0.2, 0.4, 0.6, 0.8\}$ and $\gamma = \{10^3, 10^2, 10^1, 10^0, 10^{-1}, 10^{-2}, 10^{-3}\}$ by following the previous work (Courty et al. 2017; Blondel, Seguy, and Rolet 2018). Finally, we evaluated the total processing time of $\rho = \{0.2, 0.4, 0.6, 0.8\}$ for each γ because γ adjusts the strength of the overall regularization terms. We compared our method (ours) with the original method (origin) (Blondel, Seguy, and Rolet 2018). Although we also tested the another method (Courty et al. 2017), we excluded it from the comparison since results could not be obtained for most of the hyperparameters. This was due to the numerical instability of the Sinkhorn algorithm, as pointed out in the previous work (Schmitzer 2019). In addition, that method could not achieve group sparsity due to the choice of the regularization term (Blondel, Seguy, and Rolet 2018). Each experiment was conducted with one CPU core and 264 GB of main memory on a 2.20 GHz Intel Xeon server running Linux.

Processing Time

Figure 2 shows the processing time gain on the synthetic dataset. Our method is up to 6.8 times faster than the original method. The gain increases as the numbers of class labels and data samples increase. This is because our method efficiently skips the gradient computations corresponding to the class labels. On the other hand, the checking procedure for skipping the gradient computations may become dominant when the numbers of class labels and data samples are small. In this case, our second idea of reducing the overhead works. As a result, our method turns out to be about twice as fast as the original method even with ten class labels. We confirmed that our method without the second idea was slightly slower than the existing method when the number of class labels was 10. The result suggests that the second idea helps to reduce the overhead especially for small numbers of class labels and data samples.

Figures 3, 4, and 5 show the processing time gain on the datasets for the digit, face, and object recognition tasks. In these cases, our method is up to 8.6, 3.7, 6.2 times faster than the original method in each adaptation task. The results suggest that our method is efficient even on real-world datasets.

Intuitively, our method has a large gain when γ is large because such a setting induces a sparse transportation plan, and the inequality in Lemma 1 easily holds. However, these figures suggest that the trend of gain for each γ is quite different depending on the dataset. This is because some cases converge in a few iterations depending on the value of γ and dataset. In such cases, the gain decreases since the number of gradient computations inherently small.

Number of Gradient Computations. The aim of our idea is to skip gradient computations. Therefore, we compared the number of gradient computations for the original method and our method. Figure 6 shows the results for each ρ on MNIST-USPS dataset with $\gamma = 0.1$. Our method could reduce the number of gradient computations by up to 4.22%. When the value of ρ is large, the magnitude of group-sparse regularization terms also becomes large. Since group-sparsity is aggressively induced in such a setting, our method
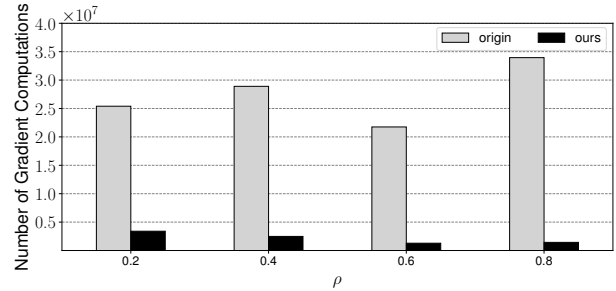

Figure 6: Numbers of gradient computations for each ρ on MNIST-USPS dataset with $\gamma = 0.1$.

Number of classes	Origin	Ours
10	2.458×10^2	2.458×10^2
20	2.530×10^2	2.530×10^2
40	2.529×10^2	2.529×10^2
80	2.529×10^2	2.529×10^2
160	2.455×10^2	2.455×10^2
320	2.530×10^2	2.530×10^2
640	1.897×10^2	1.897×10^2
1280	2.529×10^2	2.529×10^2

Table 1: Maximum objective values after convergence among all hyperparameters on the synthetic dataset.

actually skips many gradient computations for $\rho = 0.8$ in Figure 6. This figure suggests that our method efficiently skips unnecessary gradient computations.

Accuracy

We also examined the values of the objective function of Problem (4) after convergence to verify Theorem 2. Here, we will mainly show results for the maximum objective function values after convergence among all hyperparameter combinations because Problem (4) is a maximization problem. The results on the synthetic dataset are listed in Table 1. Our method achieves the same maximum objective values as those of the original method on all datasets for all hyperparameter combinations. These experimental results verify our theoretical results and suggest that our method reduces the processing time without degrading accuracy.

Conclusion

We proposed fast regularized discrete optimal transport with group-sparse regularizers. Our method exploits with two ideas. The first idea is to safely skip the gradient computations whose gradient vectors must turn out to be zero vectors. The second idea is to extract the gradient vectors that are expected to be nonzero. Our method is guaranteed to return the same value of the objective function as that of the original method. Experiments show that it is up to 8.6 times faster than the original method without degrading accuracy.

References

Alaya, M. Z.; Berar, M.; Gasso, G.; and Rakotomamonjy, A. 2019. Screening Sinkhorn Algorithm for Regularized Optimal Transport. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Bertsekas, D. P. 1999. *Nonlinear Programming*. Athena Scientific.

Blondel, M.; Seguy, V.; and Rolet, A. 2018. Smooth and Sparse Optimal Transport. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Courty, N.; Flamary, R.; and Tuia, D. 2014. Domain Adaptation with Regularized Optimal Transport. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD*.

Courty, N.; Flamary, R.; Tuia, D.; and Rakotomamonjy, A. 2017. Optimal Transport for Domain Adaptation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(9): 1853–1865.

Cuturi, M. 2013. Sinkhorn Distances: Lightspeed Computation of Optimal Transport. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Das, D.; and Lee, C. S. G. 2018a. Sample-to-Sample Correspondence for Unsupervised Domain Adaptation. *Engineering Applications of Artificial Intelligence*, 73: 80–91.

Das, D.; and Lee, C. S. G. 2018b. Unsupervised Domain Adaptation Using Regularized Hyper-Graph Matching. In *IEEE International Conference on Image Processing (ICIP)*.

Donahue, J.; Jia, Y.; Vinyals, O.; Hoffman, J.; Zhang, N.; Tzeng, E.; and Darrell, T. 2014. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. In *International Conference on Machine Learning (ICML)*.

Fujiwara, Y.; Ida, Y.; Arai, J.; Nishimura, M.; and Iwamura, S. 2016a. Fast Algorithm for the Lasso based L1-Graph Construction. *Proc. VLDB Endow.*, 10(3): 229–240.

Fujiwara, Y.; Ida, Y.; Shiokawa, H.; and Iwamura, S. 2016b. Fast Lasso Algorithm via Selective Coordinate Descent. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Gangbo, W.; and McCann, R. J. 2000. Shape Recognition via Wasserstein Distance. *Quarterly of Applied Mathematics*, 58(4): 705–737.

Gong, B.; Shi, Y.; Sha, F.; and Grauman, K. 2012. Geodesic Flow Kernel for Unsupervised Domain Adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Griffin, G.; Holub, A.; and Perona, P. 2007. Caltech-256 Object Category Dataset. Technical report, California Institute of Technology.

Gross, R.; Matthews, I.; Cohn, J.; Kanade, T.; and Baker, S. 2008. Multi-PIE. In *IEEE International Conference on Automatic Face & Gesture Recognition*.

Hull, J. J. 1994. A Database for Handwritten Text Recognition Research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5): 550–554.

Ida, Y.; Fujiwara, Y.; and Kashima, H. 2019. Fast Sparse Group Lasso. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Ida, Y.; Kanai, S.; Fujiwara, Y.; Iwata, T.; Takeuchi, K.; and Kashima, H. 2020. Fast Deterministic CUR Matrix Decomposition with Accuracy Assurance. In *Proceedings of International Conference on Machine Learning (ICML)*.

Lecun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11): 2278–2324.

Li, P.; Ni, Z.; Zhu, X.; Song, J.; and Wu, W. 2020. Optimal Transport with Dimensionality Reduction for Domain Adaptation. *Symmetry*, 12(12): 1994.

Liu, D. C.; and Nocedal, J. 1989. On the Limited Memory BFGS Method for Large Scale Optimization. *Mathematical Programming*, 45(1): 503–528.

Lu, W.; Chen, Y.; Wang, J.; and Qin, X. 2021. Cross-domain Activity Aecognition via Substructural Optimal Transport. *Neurocomputing*, 454: 65–75.

Pitié, F.; Kokaram, A. C.; and Dahyot, R. 2007. Automated Colour Grading Using Colour Distribution Transfer. *Computer Vision and Image Understanding*, 107(1-2): 123–137.

Redko, I.; Habrard, A.; and Sebban, M. 2017. Theoretical Analysis of Domain Adaptation with Optimal Transport. In *Machine Learning and Knowledge Discovery in Databases - European Conference (ECML PKDD)*.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; Berg, A. C.; and Fei-Fei, L. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3): 211–252.

Schmitzer, B. 2019. Stabilized Sparse Scaling Algorithms for Entropy Regularized Transport Problems. *SIAM Journal on Scientific Computing*, 41(3): A1443–A1481.

Venturini, L.; Baralis, E.; and Garza, P. 2017. Scaling Associative Classification for Very Large Datasets. *Jornal of Big Data*, 4(44).

Yuan, M.; and Lin, Y. 2006. Model Selection and Estimation in Regression with Grouped Variables. *Journal of the Royal Statistical Society*, 68(1): 49–67.