Convex Optimization Project Final Report

侯霁开* 贾泽宇* 李知含*
1600010681 1600010603 1600010653

January 3, 2020

1 Introduction

Optimal transport (OT) problems are an important series of problems considering the minimal cost of transportation, receiving increasing attention from the community of applied mathematics. The discrete form of optimal trnapsort problems fall into a kind of network flow problems, where the graph is restricted to be a bipartite graph and flow restrictions are freed. Cost related to a specific metric, solutions to these problems characterize the deformation between two probability distributions, and therefore the objective, called Wasserstein metric, is useful in many fields including medical image processing (feature identification), geometric learning (registration and segmentation), machine learning (feature extraction and alignment), computer vision (classification), computer graphics (blend between shapes) and even deep learning (generative adversarial networks). However, as a emerging research area, the main challenge for optimal transport problems is a lack of fast and efficient algorithm. Although there are several discrete algorithms [**Bertsekas1992**] [**Schrieber2017**] for these problems, the vast space of admissible transportation plans introduces great difficulty to computation.

In this report, we explain and report our implementation of several algorithms to discrete optimal transport problems. We include the best algorithms and answers to given questions in this report, and leave some algorithms with deficiency in either efficiency or precision in the supplementary materials. The codes and raw datium, together with the Git repository, are affliated with this report. Please refer to `Readme.md` for more details about the repository.

*The authors are arranged lexicographically.

2 Problem statement

The standard formulation of optimal transport are derived from couplings. [**Villani2009**] That is, let $(\mathcal{X}, \mu)$ and $(\mathcal{Y}, \nu)$ be two probability spaces, and a probability distribution π on $\mathcal{X} \times \mathcal{Y}$ is called *coupling* if $\operatorname{Proj}_{\mathcal{X}}(\pi) = \mu$ and $\operatorname{Proj}_{\mathcal{Y}}(\pi)\, \nu$. An optimal transport between $(\mathcal{X}, \mu)$ and $(\mathcal{Y}, \nu)$, or an optimal coupling, is a coupling minimize

$$\int_{\mathcal{X}\times\mathcal{Y}} c(x, y)\, \mathrm{d}\pi(x, y). \tag{1}$$

Optimal transport problems can be categorized according to the discreteness of μ and ν. In this report, we only consider discrete optimal tranport problems, where the two distributions are distributions of finite weighted points.

A discrete optimal transport problem can be formulated into a linear program as

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} s_{ij}, \\ \text{subject to} & \sum_{j=1}^{n} s_{ij} = \mu_i, \ (i = 1, 2, \cdots, m) \\ & \sum_{i=1}^{n} s_{ij} = \nu_j, \ (j = 1, 2, \cdots n) \\ & s \geq 0, \end{array} \tag{2}$$

where c stands for the cost and s for the transportation plan, while μ and ν are restrictions. Note that we always suppose $c \geq 0$, $\mu \geq 0$, $\nu \geq 0$ and $\sum_{i=1}^{m} \mu_i = \sum_{j=1}^{n} \nu_j = 1$ implicitly. From realistic background, c is always valued the squared Euclidean distanced or some other norms. Note that there are mn variables in this formulation, and this leads to intensive computation.

The dual problem of (2) can be written as

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{m} \mu_i \lambda_i + \sum_{j=1}^{n} \nu_j \eta_j, \\ \text{subject to} & c_{ij} - \lambda_i - \eta_j \geq 0. \ \ (i = 1, 2, \cdots, m; j = 1, 2, \cdots, n) \end{array} \tag{3}$$

Although this formulation only emploies $m + n$ variables, there are still challenges including the recovery of s from μ and ν and the great number of constraints.

Figure 1 shows an example of discrete optimal transport, where the dots represents sources and targets, and arrows represents transportations. (Tiny transportations are left out in the figure)

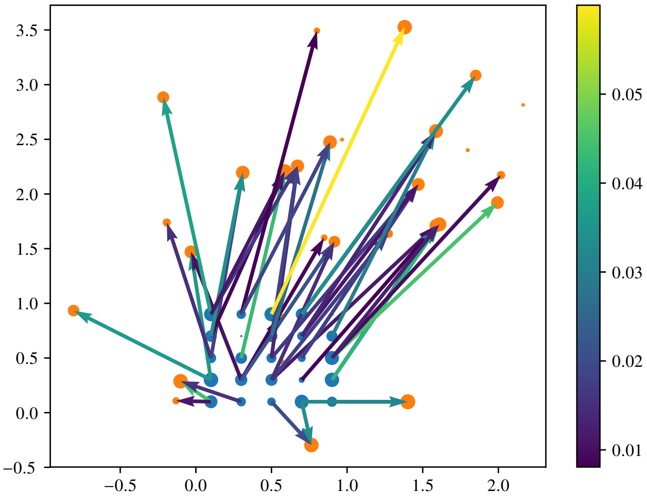

Figure 1 An example of discrete optimal transport

3 Calling solvers

According to **Question 1**, we fist solve discrete optimal transport problems by directly calling solvers MOSEK and gurobi.

Note that simplex methods are a series of methods specialized for linear programs, and therefore simplex methods are generally faster and more precise than interior point methods. However, because of special structure of this problem (the number of constraints are always fewer than that of variables), the performance of simlex methods and the interior point methods are rather close. More information can be find in Section 7.

4 First-order methods

According to **Question 2**, we implemented several first-order methods, including an ADMM for the primal problem, and a fast operator splitting method with penalty functions. Furthermore, we propose an new algorithm by combing these two algorithms.

4.1 ADMM for the primal problem

We first implement an alternative direction method of multipliers (ADMM) according to a reformulation of (2) as

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}s_{ij} + \iota_+\left(\widetilde{s}\right), \\ \text{subject to} & \sum_{j=1}^{n} s_{ij} = \mu_i,\ (i = 1, 2, \cdots, m) \\ & \sum_{i=1}^{n} s_{ij} = \nu_j,\ (j = 1, 2, \cdots n) \\ & s = \widetilde{s}, \end{array} \tag{4}$$

where ι_+ are indicator of $\mathbb{R}_+^{m\times n}$. The augmented Lagragian is

$$\begin{aligned} L_\rho\left(s, \widetilde{s}, \lambda, \eta, e\right) = & \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}s_{ij} + \iota_+\left(\widetilde{s}\right) \\ & + \sum_{i=1}^{n}\lambda_i\left(\mu_i - \sum_{j=1}^{m} s_{ij}\right) + \sum_{j=1}^{m}\eta_j\left(\nu_j - \sum_{i=1}^{n} s_{ij}\right) + \sum_{i=1}^{n}\sum_{j=1}^{m} e_{ij}\left(s_{ij} - \widetilde{s}_{ij}\right) \\ & + \frac{\rho}{2}\sum_{i=1}^{n}\left(\mu_i - \sum_{j=1}^{m} s_{ij}\right)^2 + \frac{\rho}{2}\sum_{j=1}^{m}\left(\nu_j - \sum_{i=1}^{n} s_{ij}\right)^2 + \frac{\rho}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}\left(s_{ij} - \widetilde{s}_{ij}\right)^2. \end{aligned} \tag{5}$$

One of the minimization steps can be realized explicitly by

$$\arg\min_{\widetilde{s}} L_\rho\left(s, \widetilde{s}, \lambda, \eta, e\right) = \left(s - \frac{1}{\rho}e\right)_+ \tag{6}$$

where $(\cdot)_+$ is the projection to $\mathbb{R}_+^{m\times n}$, and another can be derived from

$$\sum_{k=1}^{n} s_{ik} + \sum_{k=1}^{m} s_{kj} + s_{ij} = \frac{1}{\rho}\left(e_{ij} + \lambda_i + \eta_j - c_{ij}\right) + \mu_i + \nu_j + \widetilde{s}_{ij} \equiv r_{ij}, \tag{7}$$

which can be solved directly by its special structure. To be exact, the solution can be written explicitly as

$$s_{ij} = r_{ij} - \frac{1}{n+1}\sum_{k=1}^{n}\left(r_{ik} - \frac{1}{m+n+1}\sum_{l=1}^{m} r_{lk}\right) - \frac{1}{m+1}\sum_{k=1}^{m}\left(r_{kj} - \frac{1}{m+n+1}\sum_{l=1}^{n} r_{kl}\right). \tag{8}$$

The algorithm is shown in Algorithm 1.

Algorithm 1 ADMM for the primal problem

```
Require: $\mu$, $\nu$, $c$, step size $\rho$, scale factor $\alpha$
  $t \leftarrow 0$
  $s^{(t)}, \widetilde{s}^{(t)}, e^{(t)} \leftarrow 0$, $\lambda^{(t)} \leftarrow 0$, $\eta^{(t)} \leftarrow 0$
  while not converges do
    $s^{(t+1)} \leftarrow \arg\min_s L_\rho \left(s, \widetilde{s}^{(s)}, \lambda^{(s)}, \eta^{(s)}, e^{(s)}\right)$
    $\widetilde{s}^{(t+1)} \leftarrow \arg\min_{\widetilde{s}} L_\rho \left(s^{(t+1)}, \widetilde{s}, \lambda^{(s)}, \eta^{(s)}, e^{(s)}\right)$
    $\lambda_i^{(t+1)} \leftarrow \lambda_i^{(t)} + \alpha\rho \left(\mu_i - \sum_{j=1}^m s_{ij}\right)$
    $\eta_j^{(t+1)} \leftarrow \eta_j^{(t)} + \alpha\rho \left(\nu_i - \sum_{i=1}^n s_{ij}\right)$
    $e^{(t+1)} \leftarrow e^{(t)} + \alpha\rho \left(s - \widetilde{s}\right)$
    $t \leftarrow t + 1$
  end while
```

4.2 Fast operator splitting method with penalty functions

We then implement an accelerated operator splitting method using penalty functions. With penalty functions, the objective function can be written as $f(s) + g(s)$, where

$$f(s) = \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}s_{ij} + \pi_1 \sum_{i=1}^{n}\left(\mu_i - \sum_{j=1}^{m} s_{ij}\right)^2 + \pi_2 \sum_{j=1}^{m}\left(\nu_j - \sum_{i=1}^{n} s_{ij}\right)^2 \tag{9}$$

$$g(s) = \pi_0 \sum_{i=1}^{n}\sum_{j=1}^{m} \left(s_{ij}\right)_- \tag{10}$$

and π_1, π_2 and π_0 are penalty factors. We have found that prox_{tg} can be achieved by shrinking the negative part, that is

$$\mathrm{prox}_{tg}(s) = \max\{s, 0\} + \min\{s + t\pi_0, 0\}, \tag{11}$$

and prox_{tf} can be derived by a linear system with special structure (similar to (7)), which is

$$\begin{aligned} \mathrm{prox}_{tf}(s)_{ij} = r_{ij} & \\ & - \frac{\pi_1 t}{\pi_1 tn + 1} \sum_{k=1}^{n}\left(r_{ik} - \frac{\pi_2 t}{\pi_2 tm + \pi_1 tn + 1}\sum_{l=1}^{m} r_{lk}\right) \\ & - \frac{\pi_2 t}{\pi_2 tm + 1} \sum_{k=1}^{m}\left(r_{kj} - \frac{\pi_1 t}{\pi_1 tm + \pi_2 tn + 1}\sum_{l=1}^{n} r_{kl}\right) \end{aligned} \tag{12}$$

to be exact. We then adopt Peaceman-Rachford splitting scheme to solve this problem. Furthermore, we use Nesterov momentum in this algorithm to boost its convergence, because

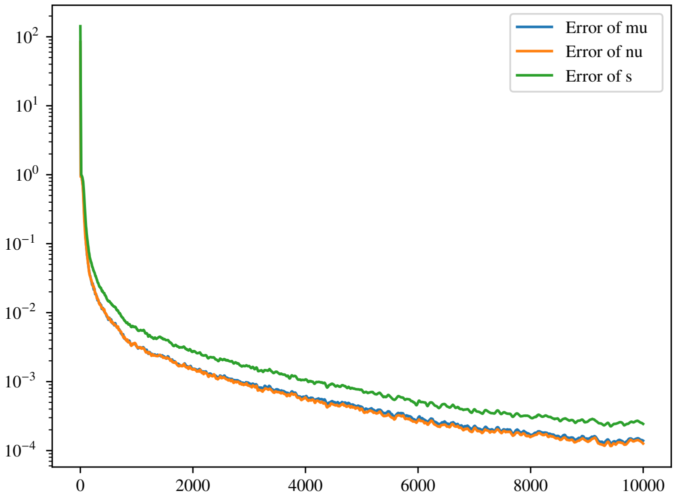

Figure 2 Error curve with respect to iterations

part of the penalty terms are quadratic. This algorithm is listed as Algorithm 2.

Algorithm 2 Operator splitting fast proximal gradient method using penalty functions

```
Require: $\mu$, $\nu$, $c$, step size $l$, penalty factors $\pi_1$, $\pi_2$, $\pi_0$.
  $t \leftarrow 0$
  $s^{(-1)}, s^{(t)} \leftarrow 0$
  while not converges do
    $s' = s^{(t)} + \frac{t-1}{t+2}\left(s^{(t)} - s^{(t-1)}\right)$
    $s^{(t+1)} \leftarrow \mathrm{prox}_{lg}\left(\mathrm{prox}_{lf}\left(s'\right)\right)$
    $t \leftarrow t + 1$
  end while
```

4.3 Proposed combined algorithm

According to some numerical experiments, we have found that Algorithm 1 suffers from difficult satisfication of constraints, while Algorithm 2 suffers from slow and hard convergence. Figure 2 shows the evolution of error of μ and ν, (see Section 7 for details) where the error fails in reaching 10^{-4} after about 10000 iterations in 44 s, while MOSEK solver only takes about 1 s. The deficiency of Peaceman-rachford splitting scheme may account for the hard convergence of Algorithm 2.

Inspired by this phenomenon, we propose a new Algorithm (**Algorithm 1+2**) by combing these two algorithms: we first perform Algorithm 1 and stop it when the constraints are not satisfied very well, (generally when the error of μ and ν reaches 10^{-3}), and then perform

Algorithm 2 to decrease the the error of μ and ν. Experiments tell Algorithm 1+2 have better efficiency and precision than the two original algorithms.

5 Transportation simplex method

According to **Question 3**, we implement the transportation simplex method mentioned in [**Schrieber2017**]. This method is similar to the ordiniary simplex method for linear programs. However, it utilizes the advantages of optimal transport problems, that is, the graph of transportation is bipartite.

5.1 Description

In the transportation simplex method, each solution to the problem corresponds to a weighted bipartite graph G as

$$G = (V, E) \qquad V = V_1 \cup V_2 \tag{13}$$

where vertices set V_1 contains all the sources, and vertices set V_2 contains all the targets. For one feasible solution s_{ij} satisfing all the constraints, we can construct the graph as follows:

$$\begin{gathered} E = \left\{(i, j) \middle| i \in V_1, j \in V_2, s_{ij} > 0\right\} \\ w_{ij} = s_{ij} \end{gathered} \tag{14}$$

where w_{ij} represents the weight of edge (i, j).

Suppose s_{ij} is an optimal solution, we may discover some properties of G:

(1) G does not have a loop;

(2) G is bipartite, which means that for all $e \in E$, e must be some (i, j) with $i \in V_1, j \in V_2$;

(3) For all $i \in V_1$, $\sum_{j \in N(i)} w_{ij} = \mu_i$;

(4) For all $j \in V_2$, $\sum_{i \in N(j)} w_{ij} = \nu_j$.

The last three property will be satisfied as long as s is feasible.

In total, the transportation simplex method can be divided into two stages:

(1) Find a basis solution satisfying all the constraints;

(2) Update the basis solution until it is optimal.

Details of each part are included in the following two subsections.

5.2 Search of basis solution

To search for a feasible solution, first we choose a vertex i in V_1 and a vertex j in V_2 and assign the maximal possible value transporting from i to j. We add a weighted edge with the maximal value from i to j and abandon one of them which has become zero. Next another vertex is chosen from U if i is abandoned or V if j is abandoned. Previous process is repeated until all the vertices are abandoned.

By this approach, we obtain a feasible solution. We can use this solution as the basis solution.

If the graph G we build based on π_{ij} is not connected, we need to add some edges in E to until G is a tree. In each addition of edge, we need to make sure that G is still acylic.

5.3 Update of solution

Given a solution s_{ij}, first we calculate the dual variable u_i (corresponding to μ_i) and v_j (corresponding to ν_i). The way to find dual variable is based on

$$u_i + v_j = 0 \tag{15}$$

where $(i, j) \in E$.

Note that if G is a tree, all the u_i and v_j are fixed as long as the initial value (one of u_i or v_j) is fixed. Consequently, $u_i + v_j$ is a fixed value for all i, j, no matter how the initial value is choosed. Thus we can calculate $c_{ij} - u_i - v_j$ without ambiguity.

If the dual variables of solution s_{ij} satisfy

$$c_{ij} - u_i - v_j \geq 0 \tag{16}$$

where $i = 1, 2, \cdots, m$ and $j = 1, 2, \cdots, n$, then our solution is optimal.

If there exists i, j not satisfying the previous inequality, we can find i, j so that $c_{ij} - \mu_i - \nu j$ is has a largest negative part. Then, we add a path from i to j in G. In this way, an even-length loop is created in the graph. We shift the maximal amount of mass possible along this cycle, that is, the mass is alternatively added to and subtracted from consecutive transports. After shifting, we may remove the edge whose weight vanishes, and we get a new transportation graph.

We use this update until $c_{ij} - u_i - v_j \geq 0$ for all i, j. Then we tell the solution corresponding to the graph is the optimal solution.

During our implementation, we find that the description of this method in [**Luenberger2008**] and [**Schrieber2017**] is not precise enough. In the reference, a process to find i, j so that $c_{ij} - \mu_i - \nu_j$ is "the most negative" is mentioned. However, we encounter some special cases that multiple minima exist. In this case, the original algorithm may fail to find a loop, which leads to a fatal failure. To tackle this problem, we propose a slight modification by record the deleted edge during the update process and remove it from the search list. Note that this slight change would not interfere the convergence of this algorithm, since there is no transportation along the edge as long as it has been deleted. However, this modification is critical to accuracy and stability.

5.4 Discussion

The algorithm is listed as Algorithm 3.

Algorithm 3 Transportation simplex method

```
Require: $m, n, \mu, \nu, c$
  Step1:
  Find a basis solution $sol$
  Build graph $G = (V, E)$ based on $sol$
  Step2:
  Calculate the dual variable $u_i$ and $v_j$ corresponding to $\mu_i$, and $\nu_j$ respectively
  if there exists some $i, j$, such that $c_{ij} - mu_i - nu_j < 0$ then
    Add $(i, j)$ in $E$ and find a loop $l$ in G
    Update weight of edges in $l$ so that the transportation in $l$ is optimal
    Delete vanished edge in $E$
    Update the solution $sol$
    Run Step2 again
  end if
  Translate $G$ and $sol$ into the transport plan
```

Unlike other optimization method, simplex method has one distinguishing character. That is, this method could always find the optimal solution, while some other continuous approximation algorithm could only converge to the optimal solution after iterations, which introduces a hard trandoff between time and precision. In our numerical experiments, the unique factor influencing the accuracy of the solution is found to be the machine precision.

Furthermore, in each update iteration, the loss will always decrease. Figure 3 shows the changing loss in iterations, where the loss decrease quickly with stability.

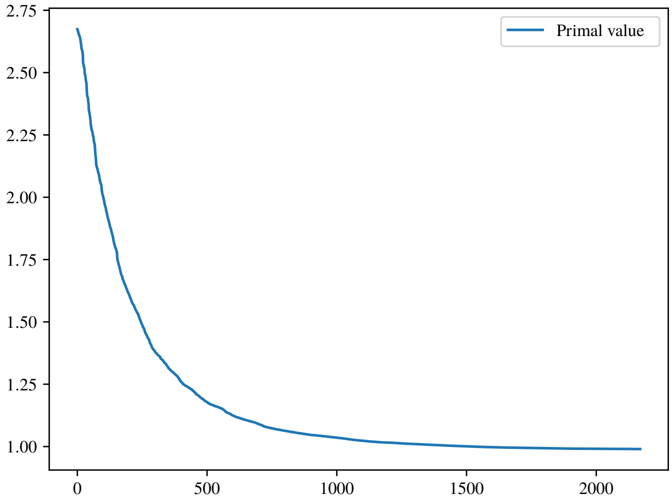

Figure 3 Curve of primal objective

6 Multiscale strategies

According to **Question 4**, we implement the multiscale method in [**Gerber2017**].

6.1 Description

From the linear program formulation (2) of optimal transport problems, the rank of linear constraints

$$\sum_{j=1}^{n} \pi_{ij} = \mu_i \ \ (i = 1, \dots, m) \tag{17}$$

$$\sum_{i=1}^{m} \pi_{ij} = \mu_j \ \ (j = 1, \dots, n-1) \tag{18}$$

is exactly $m + n - 1$. This implies that the number of non-zero elements in the solution is $n + m - 1$ in non-degenerate cases. In other words, most transport paths does not exist in the optimal transport plan. However, as a standard LP Problem, it has mn variables, which is far outweigh the number of non-zero elements of the solution.

One intuition is that we may kick absolute impossible paths off and solve the problem focused on the set of possible paths. Furthermore, we want the scale of set of possible paths to be $O(m + n)$ then we won't waste too much time solving the problem on impossible paths.

We may specify the meaning of possible path and impossible path then. Real optimal transport problems always have a good geometry structure like point cloud or image and the coefficient c_{ij} is always a function of the distance between points or pixels. As a result, paths between points with long distance always does not exist in the optimal transport plan. Due to the good geometry structure, we can regard the transport plan as transport between "big" points which are aggregation of points with short distance.

Multiscale strategies can be understood by this intuition. We aggregate small points and solve the optimal transport problem between big points to decide whether the path between small points is possible. To introduce details of multiscale strategies, we first explain the meaning of *coarsening* and *propagating*. In [**Gerber2017**] there is another concept *refining*. Actually it is a way to make "propagating" better.

A way of coarsening is to create a chain connecting the scales from fine to coarse in a multiscale fashion:

$$(X, \mu) = (X_J, \mu_J) \rightarrow (X_{J-1}, \mu_{J-1}) \rightarrow \cdots \rightarrow (X_1, \mu_1) \rightarrow \cdots \rightarrow (X_0, \mu_0) \tag{19}$$

Note that $|X_j|$ decrease as the scale decreases, and the discrete measure μ_j is a coarsification of μ at scale j, with $\mathrm{supp}(\mu_j) = X_j$. Similarly for Y and ν we obtain the chain:

$$(Y, \nu) = (Y_J, \nu_J) \rightarrow (Y_{J-1}, \nu_{J-1}) \rightarrow \cdots \rightarrow (Y_1, \nu_1) \rightarrow \cdots \rightarrow (Y_0, \nu_0) \tag{20}$$

Then we define the solution at scale j s_j as correspond optimal transport plan between μ_j and ν_j.

Propagating is a way using the solution of optimal transport problem between μ_j and ν_j to decide which path in scale $j+1$ is possible, thus giving a warm start to the problem at scale $j+1$ and reducing the number of variables.

6.2 Implementation Details

As mentioned before, one of the significant advantages of multiscale strategy is reducing the amount of variables. Therefore, we record only the possible transportation paths, in order to save all non-zeros paths instead of saving them in a sparse matrix.

6.2.1 Coarsening

We implement the algorithm and test it on point clouds and images. Approaches to coarsening on these two data sets are be slightly different. We implement coarsening on images directly by downsampling, that is, combine $k \times k$ pixels into one pixel. In practical experiments, we

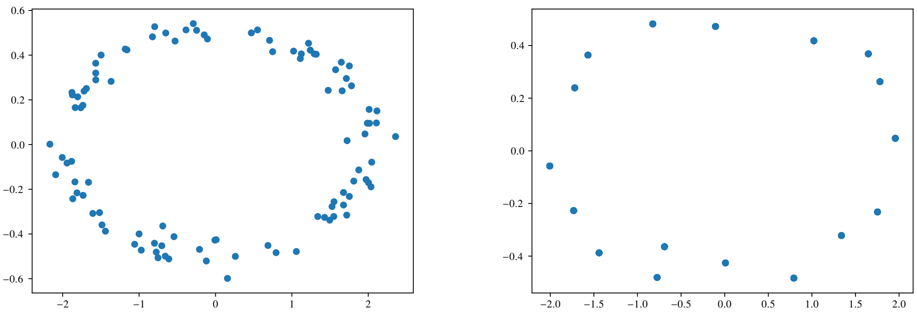

Figure 4 Division on points by a grid

choose $k = 2$.

There are various ways of coarsening on cloud points, among which are two common ways. The first is to apply K-means algorithm to the point cloud, then combine the points in the same cluster. However, we infer that using K-means here is unreasonable. In most data sets, there are no explicit cluster structure, so the output of K-means may be heavily relevant on the choice of initial point, which indicates that K-means is not capable for this task.

Conditioned this, we choose the second way that we simply divide the points by a grid. We replace the point cloud in a coordinate grid with size $N \times N$, and combine the points in the same square. That's the first step. Next, we can divide the point combined with a new gird. Noticed that we get a $N \times N$ "dot matrix" after the first step, we can use either the coarsening method of image to deal with the matrix or division. In both methods, we choose a point in each square with shortest distance to the barycenter of points lying the square as a representative point. We update a new cost matrix c_j for the squares using the cost between the representative points. Figure 4 shows an example of such sub-sampling.

Due to the fact that our data set have a good geometry structure, we inherit the metric from original space as the metric at all scale.

6.2.2 Propagating and refining

Propagation strategies is used to determine which path at the next scale should be included when the optimal transport problem at current scale have already solved. A naive approach is to include the include only paths at scale $j + 1$ whose source and targets are children of source and targets of non-zero paths at scale j. However, all paths in optimal transport plan at scale $j + 1$ can not be induced by a non-zero path at scale j, this naive approach will lead to accumulate loss of accuracy.

Due to the good geometry structure of the data set, we have a intuition that paths at scale $j+1$ should be close to paths whose source and target are derived from source and targets in the scale j. The word “close” means that the source and targets are all close.

We introduce the capacity constraint propagation to include possible paths at scale $j+1$ as much as possible. As a standard LP problem, optimal transport problem have inequality constraints $s_{ij} \geq 0$, which implies a series of trivial constraints:

$$s_{ij} \leq \min\{\mu_i, \nu_j\} \tag{21}$$

We modify those trivial constraints with a parameter λ in order to force the source to be transported to more targets, thus solution at scale j will include more possible paths at scale $j+1$:

$$s_{ij} \leq \lambda \min\{\mu_i, \nu_j\} \tag{22}$$

With these constraints, the amount of non-zero paths in the solution at scale j will increase and more possible path and scale $j+1$ will be included. The parameter λ is chosen from $[0.1, 0.9]$. The lower λ is, the higher accuracy is and the longer algorithm takes.

In numerical experiments we have realized that undersize λ may lead to a infeasible subproblem in some extreme case. So we modify the constraints into $s_{ij} \leq \lambda\mu_i$ in order to improve robustness a without loss of accuracy.

Additionally, it should be pointed out that capacity constraint is only used in propagation. At the finest scale (the original problem) the capacity are not restricted.

Refinement is another way to include more path at scale $j+1$ according to the solution at scale j. Taking advantage of the geometry structure of the data set, we introduce a kind of refinement which is called neighborhood refinement. As we had said before, paths at scale $j+1$ should be close to paths whose source and target are children of source and targets. So we can include all the paths whose sources and targets are with radius r of the sources and targets of a non-zero path at scale j into s_j, then propagate all of them to scale $j+1$.

Actually using both propagating and refining are not necessary in all situation. If we choose an enough small λ in the propagating, we may have already got enough paths at scale $j+1$ and we can save the time of refining. Refining has a time cost $O(n \log n)$ because we do not save all paths so we need to search paths in refining.

6.3 Discussion

The multiscale method is listed in Algorithm 4.

Algorithm 4 Multiscale strategy for optimal transport problems

```
Require: Source $(X, \mu)$ and target $(Y, \nu)$ as images or point clouds
  Coarsen the datium by grid or downsampling and get two chains $\{X_j, \mu_j\}_{j=0}^J$,$\{Y_j, \nu_j\}_{j=0}^J$
  Construct a path set include all paths from $(X_0, \mu_0)$ to $(Y_0, \nu_0)$
  for $j$ from 0 to $J - 1$ do
    Solve the optimal transport problem on the path set with capacity constraint at scale $j$
    Use propagation and refinement strategy to construct the path set at scale $j + 1$
  end for
  Solve the optimal transport problem without capacity constraints on the path set at scale $j$
  Construct the optimal solution $s$
```

Note that the fact that we add additional constraints to the original optimal transport problem, the subproblem we facing now is not a optimal transport problem . Furthermore, we only consider the transport on some paths so the matrix structure have been destroyed. So the solution can't be easily got by the former algorithm for optimal transport problem. In the original paper, the multiscale strategy is implemented using MOSEK and CPLEX. We implement multiscale strategy with MOSEK.

7 Numerical results and interpretations

We have tested our algorithms on three types of datasets:

(1) Randomly generated dataset;

(2) Ellipses and Caffarelli's smoothness counter examples [**Gerber2017**];

(3) DOTmark dataset [**Schrieber2017**].

The randomly generated dataset consisted of two sets of points uniformly sampled from $[0, 1] \times [0, 1]$. The weights μ and ν are randomly sampled from $[0, 1]$ and scaled to $\sum_{i=1}^m \mu_i = 1$ and $\sum_{j=1}^n \nu_j = 1$. Figure 5 shows an example.

The ellipses and Caffarelli's smoothness counter examples (Caffarelli) are generated in the nearly indentical protocol described in [**Gerber2017**]. A slight modification is made that the first ellipses is scaled by 2×0.5 and the second 0.5×2, and the two half-disks are shifted by 1 instead of 2. Figure shows an example. Note that the size of Caffarelli datasets is the number of originally sampled points, and therefore a dataset of size m actually contains $\pi m/4$ on average. Figure 6 and 7 provides examples for these two datasets.

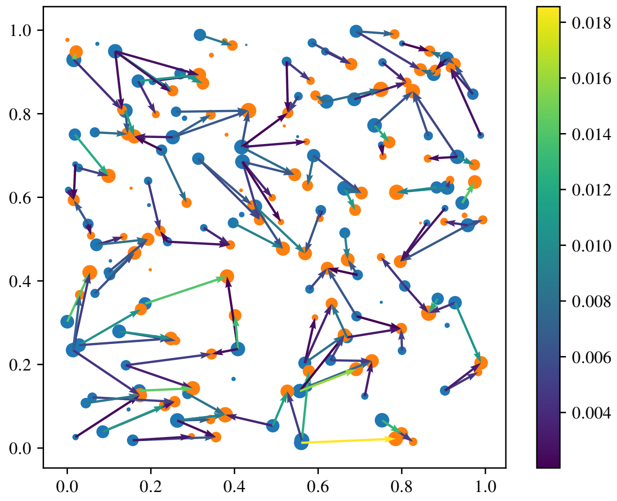

Figure 5 An transportation example on randomly generated dataset

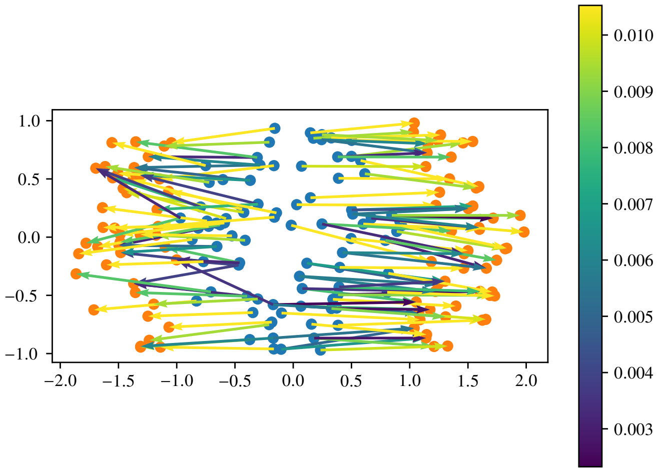

Figure 6 An transportation example on ellipses dataset

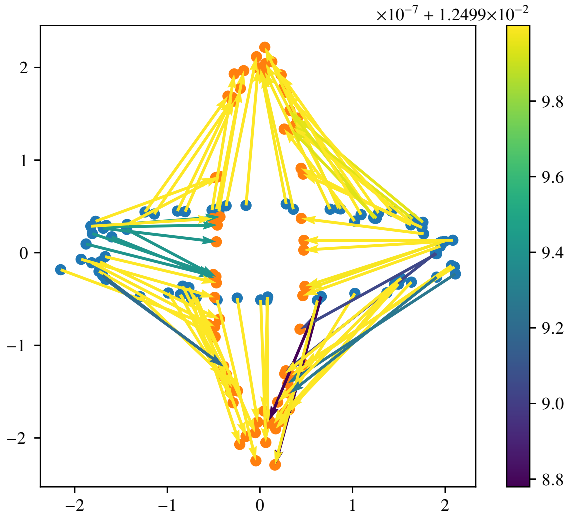

Figure 7 An transportation example on Caffarelli dataset

The DOTmark dataset is used in the described way in [**Schrieber2017**]. Time limited, we only tested a randomly chosen pair of images from each class with size 32 × 32. The classes are listed in Table 1, and Figure 8 shows the transportation in class 3.

1	WhiteNoise
2	GRFrough
3	GRFmoderate
4	GRFsmooth
5	LogGRF
6	LogitGRF
7	CauchyDensity
8	Shapes
9	ClassicImages
10	Microscopy

Table 1 Classes in the DOTmark dataset

We evaluate algorithms from three aspects:

(1) Distance to the optimal primal object, where the solution of interior point methods from MOSEK is refered as a standard;

(2) Running time;

(3) Satisfication of constraints, where we consider the 1-norm of $\mu-\sum_{j=1}^{n} s_{\cdot j}$ and $\nu-\sum_{i=1}^{m} s_{i\cdot}$ as the error to μ and ν.

We first test these algorithms on randomly generated datasets, Caffarelli datasets and

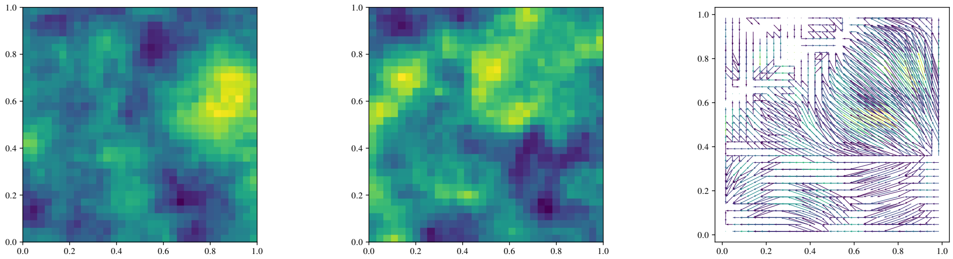

Figure 8 An transportation example on DOTmark

ellipses datasets of source size 1000 and target size 1000. Results are shown in Table 2.

		M prim.	M dual	M int	G prim.	G dual	G int	1+2	3	4
rand.	dist.	5.59e-8	4.28e-9	0.00e0	4.77e-11	4.77e-11	4.77e-11	3.46e-2	4.77e-11	2.79e-03
	time	3.66	16.21	10.21	15.1	15.84	17.6	166.68	294.19	2.36
	err. μ	1.01e-7	4.27e-17	5.73e-11	3.76e-17	6.82e-17	6.83e-17	3.33e-6	1.22e-15	1.91e-7
	err. ν	1.01e-7	7.22e-17	6.5e-11	7.00e-17	3.96e-17	4.00e-17	3.33e-6	1.27e-15	1.91e-7
Caff.	dist.	8.64e-8	5.95e-15	0.00e0	1.33e-10	3.13e-10	5.95e-15	1.06e-4	6.17e-15	2.54e-4
	time	2.04	17.37	5.46	8.73	12.52	10.43	134.72	128.97	1.37
	err. μ	8.55e-8	4.86e-17	8.96e-13	2.6e-17	4.79e-17	2.65e-12	1.12e-5	4.77e-18	3.68e-18
	err. ν	8.55e-8	5.20e-17	8.30e-13	7.48e-17	5.31e-17	7.48e-17	9.02e-6	9.62e-17	5.42e-17
ellip.	dist.	7.94e-8	6.11e-12	0.00e0	6.11e-12	6.11e-12	6.11e-12	1.18e-4	6.11e-12	2.50e-5
	time	3.15	58.74	9.53	13.93	37.18	17.07	221.35	275.89	1.2
	err. μ	8.47e-8	0.00e0	2.60e-12	0.00e0	0.00e0	0.00e0	1.48e-7	0.00e0	1.05e-16
	err. ν	8.47e-8	0.00e0	4.50e-12	0.00e0	0.00e0	0.00e0	1.49e-7	0.00e0	1.05e-16

Table 2 Numerical results for point clouds of size 1000×1000

We first test these algorithms on randomly generated datasets, Caffarelli datasets and ellipses datasets of size 250, 500, 1000, 2000 respectively. Results are shown in Table 3.

We then conduct tests on DOTmark. Results are shown in Table 4.

From the numerical results, we found that all the algorithms converge, but there are still some differences. All the algorithms has an error of μ and ν below 10^{-4}, which means the constraint are basically satisfied.

For solvers, it seems that Gurobi may always reach the truly optimal solution, while the MOSEK sometimes fails. For MOSEK, the primal simplex method are faster than the interior method and the dual simplex method. For Gurobi, simplex methods are generally faster than interior point methods. This is because an simplex methods are specified for linear programs. The bad proformance of the dual simplex may be caused by the huge amount of constraints. Gurobi being generally bettern than MOSEK because of defect of the Python interface of Gurobi.

Algorithm 1+2 is rather slow, with a moderate accuracy. This is because the size of problems tested are rather large, and ADMM, as a general algorithm, is not completely capable

		M prim.	M dual	M int	G prim.	G dual	G int	1+2	3	4
rand.	dist.	6.17e-8	1.98e-12	0.00e0	1.91e-12	1.91e-12	1.81e-12	1.48e-2	1.91e-12	1.28e-3
250	time	0.20	0.25	0.36	0.69	0.78	0.80	2.66	5.18	1.14
rand.	dist.	1.21e-7	9.74e-10	0.00e0	9.74e-10	9.74e-10	9.74e-10	4.76e-2	9.74e-10	5.54e-3
500	time	0.81	1.66	1.89	3.2	2.92	3.81	20.11	33.38	1.06
rand.	dist.	5.59e-8	4.28e-9	0.00e0	4.77e-11	4.77e-11	4.77e-11	3.46e-2	4.77e-11	2.79e-3
1000	time	3.66	16.21	10.21	15.1	15.84	17.6	166.68	294.19	2.35
rand.	dist.	3.38e-7	3.97e-8	0.00e0	6.85e-7	3.91e-7	4.66e-10	6.39e-2	3.13e0*	5.37e-3
2000	time	18.6	90.37	48.74	280.93	58.64	75.59	1172.89	1035.05	5.15
Caff.	dist.	5.34e-8	3.01e-11	0.00e0	3.01-11	3.01e-11	3.01e-11	2.01e-4	3.01e-11	1.35e-3
250	time	0.11	0.2	0.18	0.4	0.59	0.48	1.84	2.92	0.3
Caff.	dist.	8.49e-8	5.45e-13	0.00e0	5.45e-13	5.44e-13	5.44e-13	1.30e-4	5.45e-13	5.96e-4
500	time	0.45	1.71	1.16	1.86	2.52	2.11	14.31	15.16	0.64
Caff.	dist.	8.64e-8	5.95e-15	0.00e0	1.33e-10	3.12e-10	5.95e-15	1.06e-4	6.16e-15	2.54e-4
1000	time	2.04	17.38	5.46	8.73	12.51	10.43	134.72	128.97	1.37
Caff.	dist.	1.36e-7	1.54-10	0.00e0	5.30e-13	2.90e-10	2.93e-12	9.23e-5	2.59e-3*	3.68e-4
2000	time	10.42	136.17	25.99	38.58	64.5	47.21	910.79	717.34	2.43
ellip.	dist.	6.02e-8	5.11e-11	0.00e0	5.11e-11	5.11e-11	5.11e-11	1.95e-4	5.11e-11	5.28e-5
250	time	0.18	0.48	0.31	0.67	1.32	0.77	2.89	5.08	2.29
ellip.	dist.	7.79e-8	1.65e-10	0.00e0	1.65e-10	1.65e-10	1.64e-10	2.08e-4	1.65e-10	8.89e-6
500	time	0.73	5.74	1.61	3.08	6.52	3.64	31.2	35.11	0.64
ellip.	dist.	7.95e-8	6.11e-12	0.00e0	6.11e-12	6.11e-12	6.11e-12	1.17e-4	6.11e-12	2.50e-5
1000	time	3.15	58.74	9.53	13.94	37.19	17.01	221.35	275.87	1.2
ellip.	dist.	1.10e-7	4.07e-14	0.00e0	4.08e-14	4.08e-14	4.08e-14	7.91e-5	3.19e-3*	1.58e-5
2000	time	15.61	464.57	42.1	285.01	1049.3	74.86	1526.3	1045.14	1.97

*: Transportation simplex method fails to converge in 20000 iterations

Table 3 Numerical results for point clouds of different sizes

class		M prim.	M dual	M int	G prim.	G dual	G int	1+2	3	4
1	dist.	5.22e-8	1.02e-9	0.00e0	1.02e-9	1.02e-9	1.02e-9	6.38e-2	1.02e-9	1.54e-3
	time	5.12	12.84	10.8	15.43	13.06	18.98	211.32	396.68	0.75
2	dist.	1.93e-8	1.18e-10	0.00e0	1.18e-10	1.19e-10	1.19e-10	5.64e-2	1.18e-10	1.54e-4
	time	5.04	19.83	11.03	15.43	13.27	19.23	212.58	440.19	0.82
3	dist.	1.91e-8	9.92e-13	0.00e0	9.93e-13	9.93e-13	9.93e-13	6.49e-3	9.93e-13	1.80e-3
	time	4.92	27.84	10.84	15.26	18.39	19.4	230.33	458.15	0.86
4	dist.	4.36e-8	4.71e-13	0.00e0	4.71e-13	4.71e-13	4.71e-13	1.82e-3	4.71e-13	2.69e-4
	time	4.92	29.11	12.49	15.2	24.87	19.79	221.99	462.48	0.92
5	dist.	7.81e-9	5.56e-12	0.00e0	5.56e-12	5.57e-12	5.56e-12	1.75e-3	5.56e-12	1.09e-3
	time	5.59	35.41	10.96	15.14	24.4	19.54	214.4	562.46	0.81
6	dist.	3.79e-8	8.85e-10	0.00e0	8.86e-10	8.86e-10	8.86e-10	4.20e-3	8.86e-10	1.36e-2
	time	5.01	25.06	10.89	15.32	19.46	19.48	217.84	536.3	0.87
7	dist.	3.69e-8	3.03e-11	0.00e0	3.03e-11	3.03e-11	3.03e-11	1.86e-3	3.03e-11	1.23e-4
	time	4.9	41.34	10.72	14.98	33.1	19.18	221.53	421.21	0.90
8	dist.	4.07e-8	5.31e-10	0.00e0	5.31e-10	5.31e-10	5.31e-10	4.25e-2	5.31e-10	1.64e-1
	time	2.69	3.59	4.02	12.95	12.90	13.50	169.20	405.04	0.36
9	dist.	3.84e-9	2.81e-12	0.00e0	2.81e-12	2.81e-12	2.81e-12	7.06e-3	2.81e-12	3.64e-2
	time	5.23	24.96	10.65	15.32	16.87	19.91	228.10	494.80	0.87
10	dist.	3.71e-8	4.00e-9	0.00e0	4.00e-9	4.00e-9	4.00e-9	1.93e-3	4.00e-9	2.13e-3
	time	3.33	13.66	5.88	14.65	17.42	15.37	235.06	457.52	0.63

Table 4 Numerical results on DOTmark

of this problem.

Algorithm 3 has a high accuracy and precision, but it is still very slow. Actually, the dynamic feature of Python accounts for this failure, which makes discrete algorithms, especially graph algorithms very slow. Actually, programs in [**Schrieber2017**] are implemented by Java, which has better proformance in discrete algorithms.

Algorithm 4 is very fast, but its precision is not very high, this is because the multiscale algorithm itself is an approximate algorithm.

All the codes are implemented in Python. These results are carried out on a computer with Intel Core i7-6500U (4 threads) and 7890 MiB RAM. The operating system is Arch Linux 4.14.10 64-bit and the Python environment is provided by Anaconda 4.3.30. Further environment setting can be find in `environmnet.yml`.

8 Conclusion and outlook

As for optimal transport problem, our work in this semester ends up here. For these methods we have implemented, we have the following outlooks.

For first order methods, we hope to implement the interior point algorithm, together with Douglas-Rachford splitting in Algorithm 2. In addition, we also want to test out second-order algorithms like Newton methods and semi-smooth Newton methods.

For the transportation simplex method, we hope to refine the quality of basis solution to let this algorithm become faster. Besides, the way to find dual variable is very costly, if we could simplify this process, it would be much faster. Additionally, if the time have permitted, we might have tried implementing this algorithm in C++ in order to avoid intensive discrete operations in Python.

For multiscale strategies, we hope to find a theoretical proof for the convergence. Besides, we wish to refine our propagation process, which could improve the robustness by keeping the subproblem feasible even with extreme condition such as when capacity constraints is smaller.

For numerical experiments, we hope to test all algorithms on the whole DOTmark to have a more precise investigation on the precision and efficiency of all these algorithms.

As we discussed before, transportation problem is basis for many high level applications. However, solving the problem is still tough even with these brilliant methods. We also hope to find more time and space efficient methods. This will be left for future research.

9 Acknowledgement

We would like to thank Samuel Gerber in the end, for his clarification of details about the article [**Gerber2017**].

Supplementary Materials

侯霁开 贾泽宇 李知含
1600010681 1600010603 1600010653

January 3, 2020

10 Other first-order methods

Besides three algorithms mentioned in Section 4, we have also tried some other first-order methods.

10.1 ADMM for the dual problem

We have implemented an alternative direction method of multipliers (ADMM) to the dual problem according to a reformulation of (3)

$$\begin{array}{ll} \text{minimize} & -\sum_{i=1}^{m}\mu_i\lambda_i - \sum_{j=1}^{n}\nu_j\eta_j + \iota_+(e), \\ \text{subject to} & c_{ij} - \lambda_i - \eta_j - e_{ij} = 0. \ (i = 1, 2, \cdots, m; j = 1, 2, \cdots, n). \end{array} \tag{23}$$

The augmented Lagragian is

$$\begin{aligned} L_\rho(\lambda, \mu, e, d) = & -\sum_{i=1}^{m}\mu_i\lambda_i - \sum_{j=1}^{n}\nu_j\eta_j + \iota_+(e) \\ & + \sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij}\left(c_{ij} - \lambda_i - \eta_j - e_{ij}\right) \\ & + \frac{\rho}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}\left(c_{ij} - \lambda_i - \eta_j - e_{ij}\right)^2. \end{aligned} \tag{24}$$

The minimization of e can be done directly by solving for zero gradient and projection, while the minimization of λ and μ can be done by solving for zero gradient. (Actually this system has one degree of freedom, which can be fixed by letting $\sum_{j=1}^{n}\eta_j = 0$ and this does not influence the result) The algorithm is listed as Algorithm 5. Solution s can be recovered by $s = -d$ from KKT conditions.

Algorithm 5 ADMM for the dual problem

```
Require: $\mu$, $\nu$, $c$, step size $\rho$, scale factor $\alpha$
  $t \leftarrow 0$
  $\lambda^{(t)} \leftarrow 0, \eta^{(t)} \leftarrow 0, s^{(t)}, e^{(t)}, d^{(t)} \leftarrow 0$
  while not converges do
    $\lambda^{(t+1)}, \eta^{(t+1)} = \arg\min_{\lambda,\eta} L_\rho\left(\lambda, \mu, e^{(t)}, d^{(t)}\right)$
    $e^{(t+1)} = \arg\min_e L_\rho\left(\lambda^{(t+1)}, \eta^{(t+1)}, e, d^{(t)}\right)$
    $d_{i,j}^{(t+1)} \leftarrow d_{ij}^{(t)} + \alpha\rho\left(c_{ij} - \lambda_i - \eta_j - e_{ij}\right)$
    $e^{(t+1)} \leftarrow e^{(t)} + \alpha\rho\left(s - \widetilde{s}\right)$
    $t \leftarrow t + 1$
  end while
  $s^{(t)} \leftarrow -d^{(t)}$
```

However, because this algorithm introduces more variables ($3mn+m+n$) and the constraints are not introduced explicitly, it suffers from heavy computation and slow convergence.

10.2 Approximate augmented Lagragian method for the primal problem

We have also tried augmented Lagragian method (ALM) to the primal problem directly. The augmented Lagragian is

$$
\begin{aligned}
L_\rho(s, \lambda, \eta) = & \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}s_{ij} + \iota_+(s) \\
& + \sum_{i=1}^{n} \lambda_i \left(\mu_i - \sum_{j=1}^{m} s_{ij}\right) + \sum_{j=1}^{m} \eta_j \left(\nu_j - \sum_{i=1}^{n} s_{ij}\right) \\
& + \frac{\rho}{2}\sum_{i=1}^{n}\left(\mu_i - \sum_{j=1}^{m} s_{ij}\right)^2 + \frac{\rho}{2}\sum_{j=1}^{m}\left(\nu_j - \sum_{i=1}^{n} s_{ij}\right)^2 .
\end{aligned} \tag{25}
$$

To minimize s, we adopt a projection gradient step to perform a approximate minimization. The algorithm is listed as Algorithm 6.

This algorithm suffers from the approximation step heavily, which makes the error of μ and ν hard to reach 10^{-4}.

Algorithm 6 Approximate ALM for the primal problem

```
Require: $\mu$, $\nu$, $c$, step size $\rho$, scale factor $\alpha$, gradient step size $l$
  $t \leftarrow 0$
  $s^{(t)} \leftarrow 0, \lambda^{(t)} \leftarrow 0, \eta^{(t)} \leftarrow 0$
  while not converges do
    $s' = s^{(t)} - l\nabla_s L_\rho\left(s^{(t)}, \lambda^{(t)}, \eta^{(t)}\right)$
    $s^{(t+1)} = \iota_+(s')$
    $\lambda_i^{(t+1)} \leftarrow \lambda_i^{(t)} + \alpha\rho\left(\mu_i - \sum_{j=1}^m s_{ij}\right)$
    $\eta_j^{(t+1)} \leftarrow \eta_j^{(t)} + \alpha\rho\left(\nu_i - \sum_{i=1}^n s_{ij}\right)$
    $t \leftarrow t + 1$
  end while
```

11 Other implementation of transportation simplex method

As mentioned in Section 5, all the algorithms are implemented in Python, which introduces a huge disadvantages for the transportation simplex method. This is because the dynamic feature of Python, which results in slow discrete operations, especially graph algorithms. Therefore, we empoly the graph package NetworkX in order to realize some graph algorithm, and we call this implementation **Algorithm 3N**. However, this algorithm is even slower than Algorithm 3, this is because the package does not include BFS algorithms, which is the key to the whole algorithm.

12 Multiscale Strategies with ADMM

In addition to Section 4 and 6, we give a new algorithm combine the multiscale strategy and ADMM to accelerate ADMM.

In the capacity constraint propagation, we add upper bound for every variable π_{ij}. Note that a variable equal to 0 can also be saved as a variable with a upper bound 0.

We save all the paths in a $m \times n$ matrix and use another matrix to save the upper bound of each variable. In the primal ADMM algorithm, we use projection step (6) to deal with the constraints s_{ij}. Now we can use the same method to deal with the upper bound. We modify the primal ADMM algorithm slightly into a ADMM algorithm for box constraint optimal transport problem, and combine the algorithm with the multiscale strategy we have implemented before.

Notice that many variables with upper bound 0 should be 0 in the optimal solution, so project these variables to 0 may accelerate the speed of convergence, although each iteration takes the same period of time.

We save the optimal plan in a matrix because we want to use the former ADMM algorithm, so the memory cost of the multiscale strategy is not reduced. However, we have seen in the experiment result that the time cost of the multiscale algorithm are lower.

This algorithm with ADMM is listed as Algorithm 7.

Algorithm 7 Multiscale strategy with ADMM

```
Require: Source $(X, \mu)$ and target $(Y, \nu)$ as images or point clouds
  Coarsen the datium by grid or downsampling and get two chains $\{X_j, \mu_j\}_{j=0}^{J}$,$\{Y_j, \nu_j\}_{j=0}^{J}$
  Construct a path set include all paths from $(X_0, \mu_0)$ to $(Y_0, \nu_0)$
  for $j$ from 0 to $J - 1$ do
    Solve the optimal transport problem with box constraints at scale $j$ by ADMM for the primal problem
    Use capacity constraint propagation and refinement strategy to compute the upper bound matrix (box constraints) of paths at scale $j + 1$
  end for
  Solve the optimal transport problem without capacity constraints on the path set at scale $j$
  Construct the optimal solution $s$
```

We have tested this algorithm in DOTmark dataset. We stop the ADMM algorithm if error of μ and ν are below 10^{-4}. The results is shown in Table 5. The number of iterations refers to the optimization of the coarest scale.

class	1	2	3	4	5	6	7	8	9	10	average
Algorithm 1	9794	10805	10359	11867	11506	10439	13187	10827	10646	12749	11217.9
Algorithm 7	8549	11371	10973	9899	9118	9826	10677	10882	9848	10504	10164.7

Table 5 Number of iterations on DOTmark dataset

From the result, we conclude that Algorithm 7 indeed accelerates ADMM.

13 Entropy regularized methods

Entropy regularization [**Benamou2015**] is an important method in solving large scale optimization problems, which makes the optimization easier and also leads to some fast algorithms. The entropy regularization term is

$$R(s) = \sum_{i=1}^{n} \sum_{j=1}^{m} s_{ij} \left(\ln s_{ij} - 1 \right). \tag{26}$$

When the regularization coefficient $\gamma \longrightarrow +\infty$, the solution $s^\star \longrightarrow \nu^{\mathrm{T}}\mu$; and when $\gamma \longrightarrow 0$, $s^\star$ tends to the real solution, as shown in Figure 9, where γ is valued 1e1, 3e0, 1e0, 3e-1, 1e-1, 3e-2, 1e-2, 3e-3 respectively.

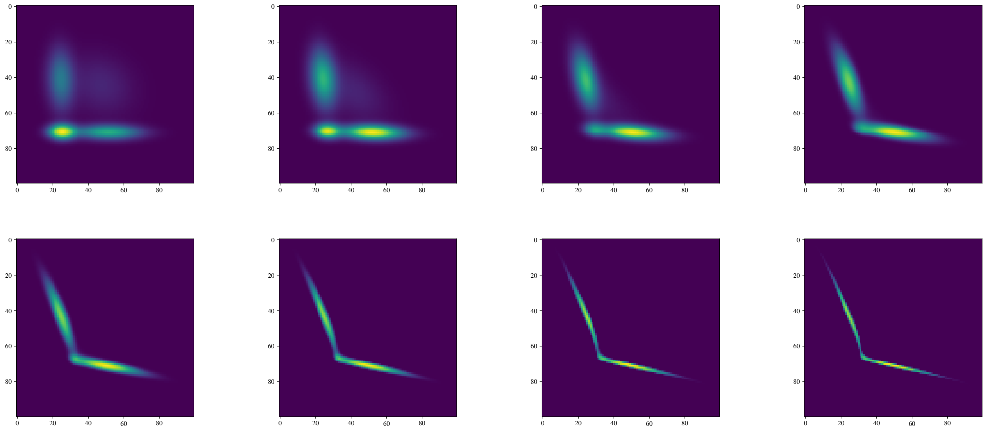

Figure 9 The influence of the regularization coefficient γ

Using entropy regularization, we may modify Algorithm 1 to a entropy regularized version **(Algorithm 1R)**. The augmented Lagragian is

$$\begin{aligned} L_{\rho\gamma}\left(s, \widetilde{s}, \lambda, \eta, e\right) = & \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}s_{ij} + \iota_{+}\left(\widetilde{s}\right) + \gamma R\left(\widetilde{s}+\delta\right) \\ & + \sum_{i=1}^{n}\lambda_i\left(\mu_i - \sum_{j=1}^{m} s_{ij}\right) + \sum_{j=1}^{m}\eta_j\left(\nu_j - \sum_{i=1}^{n} s_{ij}\right) + \sum_{i=1}^{n}\sum_{j=1}^{m} e_{ij}\left(s_{ij} - \widetilde{s}_{ij}\right) \\ & + \frac{\rho}{2}\sum_{i=1}^{n}\left(\mu_i - \sum_{j=1}^{m} s_{ij}\right)^2 + \frac{\rho}{2}\sum_{j=1}^{m}\left(\nu_j - \sum_{i=1}^{n} s_{ij}\right)^2 + \frac{\rho}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}\left(s_{ij} - \widetilde{s}_{ij}\right)^2, \end{aligned} \tag{27}$$

where δ is a small number (valued 10^{-6} in numerical experiments) to increase numerical stability. However, this algorithm is still rather slow because ADMM is used.

We have also implemented IPFP / Sinkhorn algorithm in [**Benamou2015**]. The algorithm is listed as Algorithm 8. However, the deficiency of this algorithm lies in numerical instability: the $\exp\left(-c/\gamma\right)$ step is rather dangerous for small γ, which makes it impossible to reach a close solution. Note that a large regularization term leads to a large error.

Algorithm 8 Sinkhorn algorithm

```
Require: $\mu, \nu, c, \gamma$
  $\xi \leftarrow \exp(-c/\gamma)$
  $t \leftarrow 0$
  $v^{(t)} \leftarrow 1$
  while not converges do
    $u_i^{(t+1)} \leftarrow \mu_i / \left(\xi v^{(t)}\right)_i$
    $v_j^{(t+1)} \leftarrow \nu_j / \left(\xi^{\mathrm{T}} u^{(t+1)}\right)_j$
    $t \leftarrow t + 1$
  end while$s_{ij}^{(t)} = u_i^{(t)} \xi_{ij} v_j^{(t)}$
```