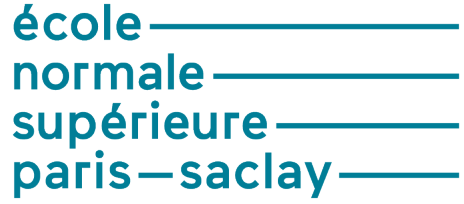

SPOT: Sliced Partial Optimal Transport

-

fast python implementation, Color transfer, and point cloud registration

Auteur:
Quentin Spinat

January 3, 2021

Contents

Abstract

This paper is the project report for the 2020 MVA course *Computational Optimal Transport* taught by Gabriel Peyré. The purpose of it is to study the paper *SPOT: Sliced Partial Optimal Transport* from Nicolas Bonneel et al.[1].

Optimal Transport is a mathematical field that focus on finding the best way to match two sets with respect to a matching cost. Here we will focus on a subproblem which consists in finding the best injective assignment of a finite set of points to a second bigger set of points. Although it is a very precise problem, it is often encountered : color transfer from an image to a bigger one, matching a small point clouds to a bigger one, ...

Nicolas Bonneel et al. proposed a method they called FIST : Fast Iterative Sliced Transport to solve this problem. It consists in solving the assignment problem considering only 1D optimal assignments, which are fast and easy to compute. Their algorithm is told to be quasilinear in time with respect to the datasets sizes.

In this project, I clarify the explanations of the original paper (which is full of typos, especially in the algorithm description), make a fast and user-friendly implementation of this algorithm in Python, and apply it to color transfer and point cloud registration.

Even though I wasn't able to parallelize my code due to python limitations, the implementation I made revealed to be really fast for color transfer and point cloud registration.

All my code is available on demand.

Keywords— Optimal Transport, Sliced Optimal Transport, Optimal assignment, Partial Transport, Stochastic Gradient Descent, Color transfer, Point Cloud Registration, Iterative Closest Point

---

Introduction

Problem

Optimal Transport is a mathematical field that focus on finding the best way to match two sets with respect to a matching cost. Here we will focus on a subproblem which consists in finding the best injective assignment of a finite set of points to a second bigger set of points. Although it is a very precise problem, it is often encountered : color transfer from an Image to a bigger one, matching a small point clouds to a bigger one.

Previous Work

Optimal transport is a computationally complex problem, often expressed as a linear program due to Kantorovich. The variables being optimized form what is known as a transport plan to describe the amount of mass from each bin of

an input (possibly high-dimensional) histogram that needs to travel to each bin of a target histogram. In this definition, it is assumed that both histograms are normalized. Directly solving this linear program can be very costly in time and memory, and when dealing with a large amount of points, algorithms become very greedy, especially because of the curse of dimensionality. Faster alternate approaches have been developed to tackle particular cases, but cannot be applied to the problem we try to overcome.

This is why the field of sliced optimal transport has emerged. It consists in projecting the point clouds onto random one-dimensional subspaces, and solving one-dimensional transport problems [Pitie et al. Rabin et al. [2], Bonneel et al. [3], [4]]. Slicing methods are very fast because of how simpler one-dimensional optimal transport is to solve. However, Sliced optimal transport is only an approximation and comes at the cost of being optimal only on the projections and not necessarily on the higher dimensional space. But optimal transport for non-normalized histograms has received little attention, even in one-dimension, which makes the sliced approach not possible for point clouds of different sizes yet.

When histograms are not normalized, the transport problem needs to be relaxed. Notably, the "unbalanced" optimal transport of Benamou et al. [5] replaces hard constraints by soft constraints in a Monge formulation to accommodate for densities of unequal masses within a PDE framework. However, it suffers from a big space complexity. Figalli introduces *partial optimal transport* that keeps all constraints of the linear program hard [6], but replaces equality with inequality constraints. The problem we are interested in is a special case of this linear program for the problem of matching point clouds.

New Contributions

Nicolas Bonneel et al. [1] proposed a new algorithm that solves one-dimensional partial optimal transport efficiently. It hence makes it possible to solve partial optimal transport in the sliced framework. They call it Sliced Partial Optimal Transport (SPOT).

Applications

In this project, we focus on two applications of this new method : color transfer problem and point cloud registration.

Color Transfer

Transferring colors between images has become a classical image processing problem. The purpose is to change the color distribution of an input image, without changing its content, to match the style of a target image. Numerous optimal transport solutions have already been developed [Bonneel et al. [4], [7]; Pitié et al. [2]; Pitié et al. [8]; Rabin et al. [9]]. In addition to their use of optimal transport, a common point to these approaches is that they consider the problem of matching normalized histograms. A consequence of that is often acknowledged as a limitation: their content should not differ. For instance, matching an image with 80% trees and 20% sky to an image containing the opposite ratio will inevitably lead

to trees becoming blue or sky becoming green. Several approaches also require the images to have exactly the same number of pixels – this is precisely the case for sliced transportation [Bonneel et al. [4]; Rabin et al. [9]].

Bonneel et al. proposed two solutions to overcome these problems, but I only focus on one of them : this solution simply enlarges the target images to give more freedom to each input pixel to be matched to target pixel values. This method is really fast and produces really good results.

Point Cloud Registration

A common problem in point cloud processing is that of registering points under a given transformation model. For instance, one tries to match a point cloud with another by supposing the transformation between them is rigid, or constrained to a similarity, or affine, or homographic, etc. A well-known algorithm to solve this problem is the Iterative Closest Point (ICP) algorithm. Given a point cloud X_0 to be matched against X_1 , this algorithm proceeds by first matching points of X_0 to their nearest neighbors in X_1 , and, given this assignment, the best transformation in the class of allowed transformations is found by minimizing an energy. The algorithm then transforms the initial point cloud using the computed transformation. The process is repeated until convergence. However, this algorithm requires points clouds to be relatively close to start with and suffers from local minima. In practice, extremely bad behaviors may arise when considering similarity transforms (rotation, translation and scaling). In that case, the lack of injectivity of the nearest neighbor map tends to estimate progressively smaller scaling factors as iterations increase, occasionally leading to a trivial zero scale solution.

This motivates the use of a metric which accounts for an injective mapping, such as the sliced metric proposed by Bonneel et al. They thus propose to replace the nearest neighbor matching by a partial sliced optimal transport matching they call *Fast Iterative Sliced Transport* (FIST).

1 Partial Transport in 1-D

This section will explain the core of this new method, which is an efficient algorithm for Partial Transport in one dimension. It relies on the fact that the nearest neighbor assignment of two point sets can be found in linear time in one-dimension. However, this nearest neighbor assignment not being injective, we need to smartly modify it. This can easily be done in quadratic time. To go further and improve the algorithm performances, we need then to introduce a way to decompose the original problem into independent easier problems, which can be solved in parallel. With this problem decomposition, the whole algorithm becomes quasilinear in time.

In all the problem we consider that X is a set of 1-D points of cardinal m, Y a set of 1-D points of cardinal $n < m$, and that X and Y are sorted by increasing value (this can be done in quasilinear time). We also write $\alpha = \frac{m}{n}$.

1.1 Nearest Neighbor Assignment

In one dimension, the nearest neighbor assignment $t : [\![1, m]\!] \to [\![1, n]\!]$ from X to Y can be found in linear time. Since $t(i) \leqslant t(i+1)$, the algorithm consists in scanning X and Y simultaneously from left to right and comparing their value.

Algorithm 1 Nearest Neighbor Assignment

Input: sorted X, Y
Output: t

$i \leftarrow 1$
$j \leftarrow 1$
while $i < m$ **do**
    if $x_i \leqslant y_j$ **then**
        $t[i] \leftarrow j$
        $i \leftarrow i + 1$
    else if $j = n$ **then**:
        $t[i] \leftarrow n$
        $i \leftarrow i + 1$
    else if $y_{j+1} < x_i$ **then**
        $j \leftarrow j + 1$
    else if $|x_i - y_j| < |x_i - y_{j+1}|$ **then**:
        $t[i] \leftarrow j$
        $i \leftarrow i + 1$
    else
        $t[i] \leftarrow j + 1$
        $j \leftarrow j + 1$
        $i \leftarrow i + 1$
    end if
end while
return t

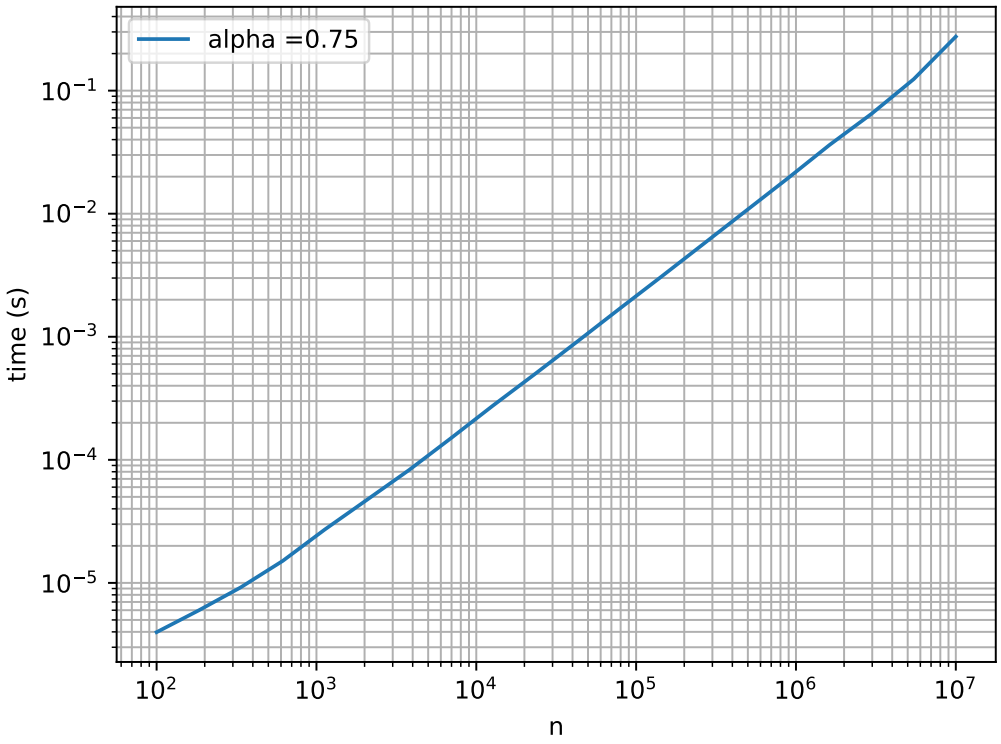

Figure 1: Mean time over 100 simulations for Nearest Neighbor Assignment with respect to n, $\alpha = 0.75$, logscale

1.2 Quadratic Time Algorithm

As the nearest neighbor assignment may not be injective, we need to change it resolving issues around points where $t[i] = t[j]$ for $i \neq j$. To do so, we scan X from left to right and consider the sub-problem of matching $X' = \{x_i\}_{i \in [\![1;m']\!]}$ with Y , progressively increasing m' from 1 to m.

To create this assignment a from t, we start by initializing $a[1] = t[1]$. Once $\{a[k]\}_{k \in [\![0;i]\!]}$ is created, there is two options. If $t[i+1] > a[i]$, we put $a[i+1] = t[i+1]$. Else, there are two possible cases : we can either put $a[i+1] = a[i] + 1$ (case 1), or put $a[i+1] = a[i]$ and translate every previous $a[k]$ from one to the left (case 2). The second option writes $a[i+1] = a[i]$ and $a[r : i] = a[r : i] - 1$, where r is such that $y_{a[r]-1}$ is the last available free spot of Y. We chose the solution that minimizes the distance between points (i.e $\sum_{k=1}^{i+1}(x_k - y_{a[k]})^2$).

Proof that the partial assignment given by this algorithm is optimal is given in appendix 4.1.

This algorithm can be optimized to run faster. First, there are trivial subcases :

- If $m = n$, we can directly put $a[i] = i$ for i from 1 to m.
- If $m = n - 1$, there exists some k to be determined such that $a[1 : k-1] = [1 : k-1]$ and $a[k : m] = [k+1 : n]$. Such k minimizes $\sum_{i=1}^{k-1}(x_i - y_i)^2 + \sum_{i=k}^{m}(x_i -$

Algorithm 2 Quadratic Partial Optimal Assignment

Input: sorted X, Y
Output: a

compute t
$a[1] \leftarrow t[1]$
for i from 1 to $m-1$ **do**
  if $t[i+1] > a[i]$ **then**
    $a[i+1] \leftarrow t[i+1]$
    update r
  else
    $w_1 \leftarrow \sum_{k=r}^{i}(x_k - y_{a[k]})^2 + (x_{i+1} - y_{a[i]+1})^2$
    $w_2 \leftarrow \sum_{k=r}^{i}(x_k - y_{a[k]-1})^2 + (x_{i+1} - y_{a[i]})^2$
    if $w_1 \leqslant w_2$ **then** ▷ Case 1
      $a[i+1] \leftarrow a[i] + 1$
    else ▷ Case 2
      $a[i+1] \leftarrow a[i]$
      $a[r:i] \leftarrow a[r:i] - 1$
      update r
    end if
  end if
end for
return a

$y_{i+1})^2$. Denoting $C = \sum_{i=1}^{m}(x_i - y_{i+1})^2$, the above minimization problem can be written $\min_k \sum_{i=1}^{k-1}(x_i - y_i)^2 + C - (xi - y_{i+1})^2$ which can be obtained within a single linear search, as C is a constant and need not be computed.

- If t is injective, we put $a = t$.

Second, we can simplify the problem before computing a.

- If there exists i such that $X[1:i] \leqslant Y[1:i]$, we can put $a[1:i] = [\![1; i]\!]$
- If there exists i such that $Y[i:n] \leqslant X[i:m]$, we can put $a[i:m] = [\![i; m]\!]$
- If p is the number of non-injective values of t, $p = card\{t[i] = t[i+1], \forall i < m\}$, then, it is enough to consider the sub-problem of matching X to $Y' = \{y_j\}_{j \in [\![max(1, t[1]-p); min(t[m]+p, n)]\!]}$

Finally, we don't have to re-evaluate the sums w_1 and w_2 each time. If we write $S_{i,1} = \sum_{k=r}^{i}(x_k - y_{a[k]})^2$ and $S_{i,2} = \sum_{k=r}^{i}(x_k - y_{a[k]-1})^2$, then in the loop i, we have $w_1 = S_{i,1} + (x_{i+1} - y_{a[i]+1})^2$ and $w_2 = S_{i,2} + (x_{i+1} - y_{a[i]})^2$. There is then three cases :

- If we put $a[i+1] > a[i] + 1$, then $S_{i+1,1} = (x_{i+1} - y_{a[i+1]})^2$ and $S_{i+1,2} = (x_{i+1} - y_{a[i+1]-1})^2$
- As long as we put $a[i+1] = a[i] + 1$, we just have $S_{i+1,1} = S_{i,1} + (x_{i+1} - y_{a[i+1]})^2$ and $S_{i+1,2} = S_{i,2} + (x_{i+1} - y_{a[i+1]-1})^2$

- In the last case (case 2), we need to re-compute almost entirely $S_{i+1,1}$ and $S_{i+2,2}$

Algorithm 3 Optimized Quadratic Partial Optimal Assignment

Input: sorted X, Y
Output: a

compute t
simplify the problem
$a[1] \leftarrow t[1]$
$S_1 \leftarrow (x_1 - y_{a[1]})^2$
$S_2 \leftarrow (x_1 - y_{a[1]-1})^2$
for i from 1 to $m-1$ **do**
    if $t[i+1] > a[i]$ **then**
        $a[i+1] \leftarrow t[i+1]$
        if $a[i+1] > a[i]+1$ **then**
            $r \leftarrow i+1$
            $S_1 \leftarrow (x_{i+1} - y_{a[i+1]})^2$
            $S_1 \leftarrow (x_{i+1} - y_{a[i+1]-1})^2$
        else ▷ r does not change
            $S_1 \leftarrow S_1 + (x_{i+1} - y_{a[i+1]})^2$
            $S_2 \leftarrow S_2 + (x_{i+1} - y_{a[i+1]-1})^2$
        end if
    else
        $w_1 \leftarrow S_1 + (x_{i+1} - y_{a[i]+1})^2$
        $w_2 \leftarrow S_2 + (x_{i+1} - y_{a[i]})^2$
        if $w_1 \leqslant w_2$ **then** ▷ case 1
            $a[i+1] \leftarrow a[i]+1$
            $S_1 \leftarrow S_1 + (x_{i+1} - y_{a[i+1]})^2$
            $S_2 \leftarrow S_2 + (x_{i+1} - y_{a[i+1]-1})^2$
        else ▷ case 2
            $a[i+1] \leftarrow a[i]$
            $a[r:i] \leftarrow a[r:i] - 1$
            update r
            $S_1 \leftarrow \sum_{k=r}^{i+1} (x_k - y_{a[k]})^2$
            $S_2 \leftarrow \sum_{k=r}^{i+1} (x_k - y_{a[k]-1})^2$
        end if
    end if
end for
return a

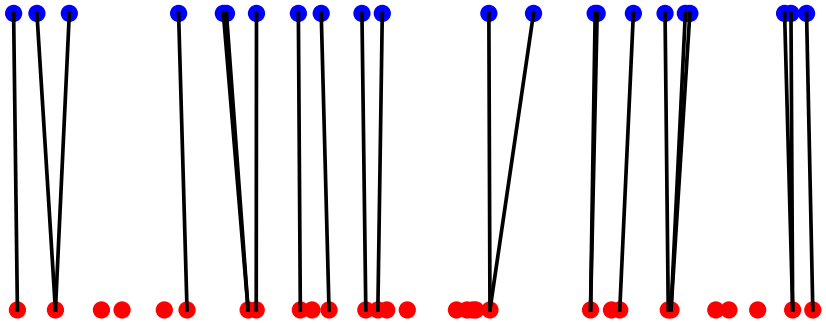

Figure 2: Nearest Neighbor Assignment

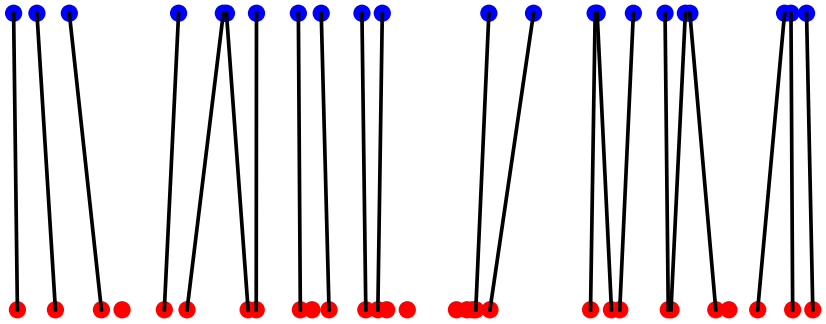

Figure 3: Partial Optimal Assignment

Surprisingly, after performances analysis, Numba accelerates loops and succeed in making the quadratic assignment linear with n for n not too big. This is better than expected, and problem decomposition may not be necessary for n smaller than one million. However it could still enables loop parallelization, and reduce complexity for bigger n.

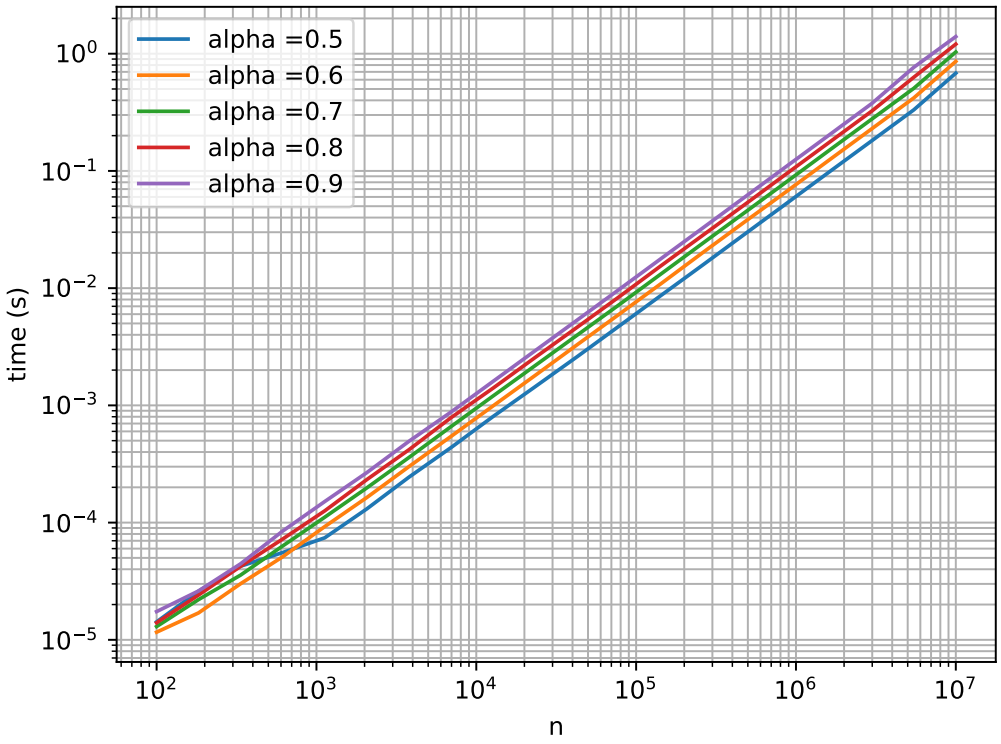

Figure 4: Mean time over 100 simulations for Quadratic Partial Optimal Assignment with respect to n, for different values of α, logscale

1.3 Quasilinear Time Problem Decomposition

In order to save more computational time, and decrease the complexity of the algorithm Bonneel et al. [1] proposed an algorithm to decompose the original problem into many easier subproblems, which can be solved independently with the previous partial optimal assignment algorithm. By doing this, it is possible to parallelize each subproblem solving and gain a consequent amount of time. However, Numba [10] the Python library for fast calculation I used, encounters problems with efficient parallelization and I wasn't able to parallelize problem solving after subproblem decomposition. An efficient implementation would require either to debug the Numba library, or to re-implement everything in a compiled language as C++.

The pseudo-algorithm provided by Bonneel et al. seemed to be false (or full of typo), and easy examples where it doesn't work are easy to find. (For example on their Fig. 3, f_{s_2} and f_{s_3} are True so problems Y_2 and Y_3 are not supposed to merge). I hence took the liberty to rewrite it differently.

The main intuition to the decomposition algorithm is that when considering a new point assignment, we need to either create a new subproblem (if $t[i+1] > a[i]$), or increase the size of the current problem. When increasing the size of the current problem, it is important to note that we just need to consider one more point on the right (case where $a[i+1] = a[i]+1$) and one more point on the left (case where $a[i+1] = a[i]$ and translation).

Specifically, we again consider an increasing sub-problem involving $\{x_i\}_{i\in[\![1;m']\!]}$ and keep track of the last available free spot s as well as the currently last value l considered by each subproblem. The sub-problem A_k thus involves $\{y_j\}_{j\in[\![s_k+1;l_k]\!]}$. We also store a flag $\{f_j\}_{j\in[\![1;n]\!]}$ where $f_j =$ false indicates that y_j has been considered by a subproblem at some point in the algorithm, or $f_j =$ true otherwise.

Algorithm 4 Decomposition of the assignment problem

Input: sorted X, Y
Output: A

compute t
initialize Boolean flags $\{f_j\}_{j\in[\![1;n]\!]}$ to true;
for m' from 1 to m **do**
  if $f_{t[m']}$ is True **then** ▷ create new subproblem
    create new subproblem $A_k = (\{x_{m'}\}, \{y_{t[m']}\})$
    $f_{t[m']} \leftarrow$ False
  else
    Consider $A_{k'}$ last subproblem
    if $t[m'] \neq t[m'-1]$ **then** ▷ expand on the right only
      $A_{k'} \leftarrow (X_{k'} \cup \{x_{m'}\}, Y_{k'} \cup \{y_{l_{k'}+1}\})$
    else ▷ expand on the right and on the left
      while $f_{s_{k'}}$ is false **do** ▷ merge with previous subproblems
        Consider $A_{k'-1}$ subproblem before $A_{k'}$
        $A_{k'-1} \leftarrow A_{k'} \cup A_{k'-1} = (X_{k'-1} \cup X_{k'}, Y_{k'-1} \cup Y_{k'})$
        $k' \leftarrow k' - 1$
      end while
      $A_{k'} \leftarrow (X_{k'} \cup \{x_{m'}\}, Y_{k'} \cup \{y_{s_{k'}}, y_{l_{k'}+1}\})$
    end if
  end if
end for
return A

It is important to note that the algorithm use the fact that X and Y are sorted, and is hence made in such a way that consecutive subproblems A_k and A_{k+1} consider consecutive values of X and increasing values of Y, which means that $l_k \leqslant s_{k+1}$.

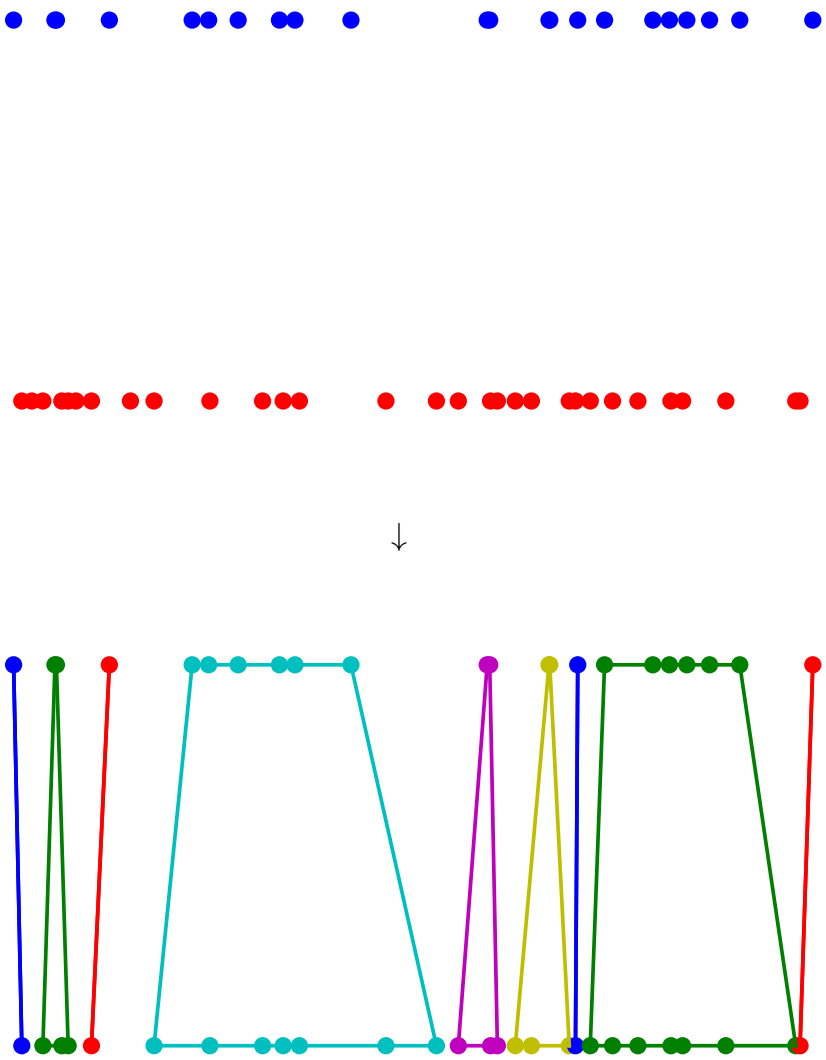

Figure 5: Assignment problem decomposition

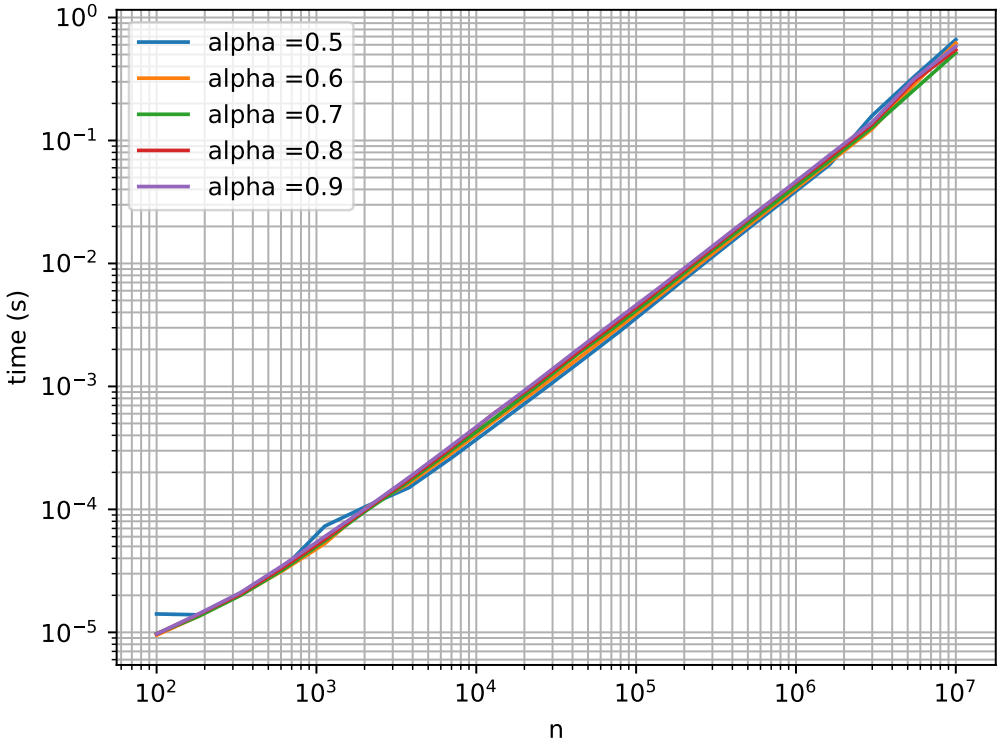

Figure 6: Mean time over 100 simulations for the decomposition of the assignment problem with respect to n, for different values of α, logscale

2 Sliced Partial Transport

While 1-d optimal transport has limited interest per se, it is the main ingredient of sliced optimal transport. Sliced optimal transport is a very fast algorithm that shares similar properties with optimal transport and works in n dimensions.

The partial 1-d optimal transport solution provided in the last section, while of quadratic complexity in the worst case, is fast (see Fig. 1, 4, 6) and is thus amenable to the computation of sliced optimal transport.

In the sliced framework, instead of minimizing directly the n-dimensional Wasserstein distance, we minimize the mean over all one-dimensional subspaces of the one-dimensional Wasserstein distance between the one-dimensional projections of the point sets :

$$\min_{\tilde{X}} \int_{S^{d-1}} W_S(Proj_\omega(\tilde{X}), Proj_\omega(Y))\, d\omega \tag{1}$$

where S^{d-1} is the (d-1)-dimensional sphere of directions.

To do so, we use a classical stochastic gradient descent with mini-batch size of one. We start by initializing $X_0 = X$. Then, at each iteration k, we choose a random direction ω and compute the gradient of $E_\omega(X_k) = W_S(Proj_\omega(X_k), Proj_\omega(Y))$ which is :

$$\nabla_X E_\omega(X_k) = Proj_\omega(X_k) - Proj_\omega(Y \circ a) \tag{2}$$

where a is the optimal partial assignment between $Proj_\omega(\tilde{X})$ and $Proj_\omega(Y)$.

We update X_k with the step:

$$X_{k+1} = X_k - \eta_k \nabla_X E_\omega(X_k)$$

With $\eta_k > 0$ the learning rate. Formally, in order to be sure of the algorithm convergence, η_k must be chosen such as $\sum_{k=0}^{+\infty} \eta_k = +\infty$ and $\sum_{k=0}^{+\infty} \eta_k^2 < +\infty$. In fact, this algorithm works well with a constant learning rate for cases in which we are interested.

At the end of the algorithm, we get X^* an approximation of the optimal partial assignment of X to Y.

3 Results

3.1 Color Transfer

Once all the previous method has been implemented, color transfer is easy to do : The algorithm is just a stochastic gradient descent described as above, with sliced partial optimal transport, and a learning step η constant equal to 1. The use of sliced partial optimal transport makes it possible to transfer color from images of different sizes. But the interesting point about that, is that it makes it possible to do a good color transfer between images that have their content not in the same proportion.

For example in Fig. 7 and 8, the input image has a lot more trees than the target image. As a consequence, if no upscaling of the target image is done, trees takes the blue color of the sky.

Figure 7: Color transfer between images using sliced partial optimal transport, input image has more trees

Figure 8: Color transfer between images using sliced partial optimal transport, input image has more trees and is bluer

In the same way, in Fig. 9, the target image has too much blue, and if no upscaling is done, almost the entire output image is blue.

Figure 9: Color transfer between images using sliced partial optimal transport, input image is bluer

However, doing too much upscaling can result in a bad output image. This can be seen in the Fig. 10, where a scale foactor of 1.1 or 1.2 seems better than 1.5, where the trees have taken the color of the mountains of the target image.

Figure 10: Color transfer between images using sliced partial optimal transport, input and target images have the same content

Every simulations were done with images of size 500×281 pixels, and with 300 iterations. It took between 1 and 5 minutes (depending on the scale factor) to run.

3.2 Point Cloud Registration

Point Cloud registration required a different method which is variation of the Iterative Closest Point Algorithm : the FIST. As stated in the introduction, this algorithm consists in doing a sliced optimal transport in one-dimension, then finding the best similarity transformation to match the transformation described by the partial assignment, and finally update the point cloud with this transformation. To find the best similarity transformation, we solve the Orthogonal Procrustes problem [11], to find the best rotation between the centered point cloud and its centered assignment, we scale it and recenter it, so it has the same scale and center as its assignment in the target point cloud.

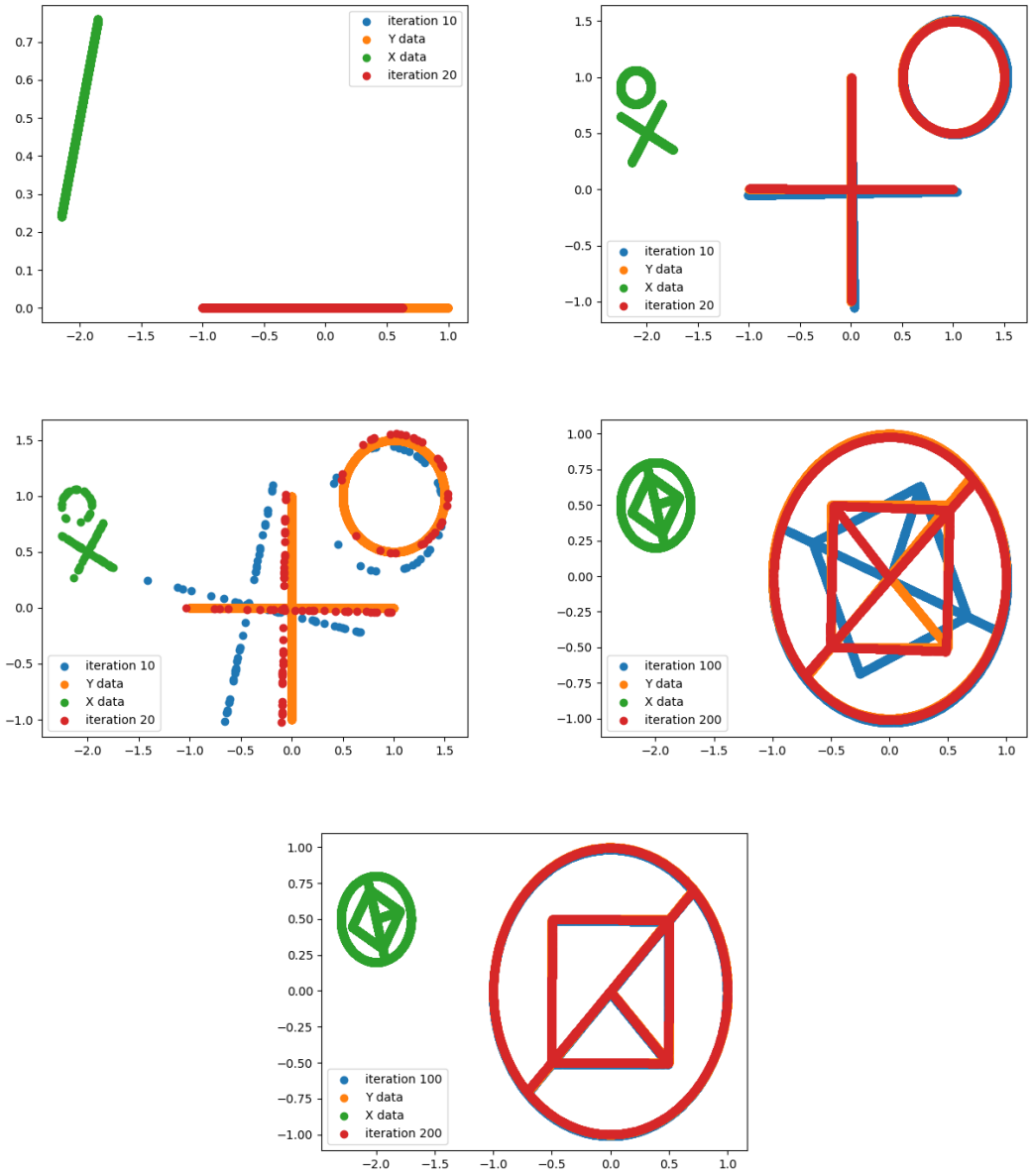

Figure 11: Point cloud Registration for different shapes

Results are impressive for shapes that have not too much symmetries. For example, for the shape with the cross and the circle, there is convergence with 20 iterations, even with only 1% of the points. However, there is never convergence with the line, no matter the number of iterations. Finally with the last shape, which is more complex, there is always convergence, but sometimes to a local minimum, and it requires at least 200 iterations to converge.

In the figure 11, the target point cloud is of size 10 000, and the input point cloud is of size 8 000, except for the third figure where the input cloud is of size 100. It takes approximately one second for 50 iterations with input and target point clouds respectively of size 10 000 and 8 000.

4 Appendix

4.1 Proof of the optimality of algorithm 2

By mathematical induction:
Let $n \in \mathbb{N}$. Let t be the nearest neighbor assignment from X to Y. Let $P(m)$ be the property : algorithm 2 gives the optimal assignment for $card(X) = m$ and $card(Y) = n$.
Initial step: It is trivial for m=1
Inductive Step: Assume that $P(m)$ is true. We want to show that $P(m+1)$ is true. Tanks to $P(m)$, we have $a_m : [\![1; m]\!] \to [\![1; n]\!]$ the optimal partial assignment from $\{x_i\}_{i \in [\![1;m]\!]}$ to Y.
If $a_m[m] < t[m+1]$, then a_{m+1} such that $a_{m+1}(i) = a_m(i)$ if $i \leqslant m$ and $a_{m+1}(m+1) = t(m+1)$ is clearly optimal.
If $t[m+1] \leqslant a_m[m]$, then, x_{m+1} cannot be optimally assigned to any y_l with $l > a_m[m] + 1$. Indeed, as the nearest neighbor of x_{m+1} is (strictly) before $y_{a_m[m]+1}$, any other assignment of x_{m+1} would have a higher cost.
Moreover, an optimal 1-D assignment a has no intersections. That means that there is no $x_i < x_j$ such that $y_{a[j]} < y_{a[i]}$. Therefore x_{m+1} cannot be assigned to y_l with $l < a_m[m]$, and if x_{m+1} is assigned to $y_{a_m[m]}$, then $a_{m+1}[r : m] = a_m[r : m] - 1$.
As a consequence, is only two cases left, which are the case one and the case 2 of the algorithm. The optimal partial assignment a_{m+1} is that which has the smaller cost between case 1 and case 2. Therefore a_{m+1} as described by the algorithm is optimal.
Conclusion: $P(m)$ is true for $m \in [\![1; n]\!]$. Therefore algorithm 2 is optimal for all $n \in \mathbb{N}$ and for all $m \in [\![1; n]\!]$.

Conclusion

To Summarize, in this project I have :

- Made a fast python implementation of Sliced Partial Optimal Transport. This method is fast and efficient in time and memory. I have implemented the problem decomposition but I am not able to parallelize it easily because of a bug in the numba library. To overcome this problem, one should either implement it in C++, debug numba, or use cython instead of numba.
- Implemented a color transfer method that works with images that have content with different proportions. It works well and it is fast. However it is not always better to upscale the target image, as it can lead to a faded image.
- Implemented the FIST algorithm for point cloud registration. It is a really fast method for matching point clouds. I have tested it in 2D but not in 3D, even though it would be easy to do. This method could be improved by adding a way not to stay in a local minimum.

References

[1] Nicolas Bonneel and David Coeurjolly. Spot: Sliced partial optimal transport. *ACM Transactions on Graphics (SIGGRAPH)*, 38(4), 2019.

[2] Francois Pitie, Anil C Kokaram, and Rozenn Dahyot. N-dimensional probability density function transfer and its application to color transfer. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1434–1439. IEEE, 2005.

[3] Julien Rabin, Gabriel Peyré, Julie Delon, and Marc Bernot. Wasserstein barycenter and its application to texture mixing. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 435–446. Springer, 2011.

[4] Nicolas Bonneel, Julien Rabin, Gabriel Peyré, and Hanspeter Pfister. Sliced and radon wasserstein barycenters of measures. *Journal of Mathematical Imaging and Vision*, 51(1):22–45, 2015.

[5] Jean-David Benamou. Numerical resolution of an "unbalanced" mass transport problem. *ESAIM: Mathematical Modelling and Numerical Analysis-Modélisation Mathématique et Analyse Numérique*, 37(5):851–868, 2003.

[6] Alessio Figalli. The optimal partial transport problem. *Archive for rational mechanics and analysis*, 195(2):533–560, 2010.

[7] Nicolas Bonneel, Kalyan Sunkavalli, Sylvain Paris, and Hanspeter Pfister. Example-based video color grading. *ACM Trans. Graph.*, 32(4):39–1, 2013.

[8] François Pitié, Anil C Kokaram, and Rozenn Dahyot. Automated colour grading using colour distribution transfer. *Computer Vision and Image Understanding*, 107(1-2):123–137, 2007.

[9] Julien Rabin, Julie Delon, and Yann Gousseau. Regularization of transportation maps for color and contrast transfer. In *2010 IEEE International Conference on Image Processing*, pages 1933–1936. IEEE, 2010.

[10] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, New York, NY, USA, 2015. Association for Computing Machinery.

[11] Peter H Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966.