Sorbonne Université

Laboratoire d'Informatique de Paris 6

Mémoire

Spécialité Informatique

par

Joseph Budin

Fast Auction-based Computation of Wasserstein Barycenters of Persistence Diagrams

Confidential

Please, do not share this document without the explicit consent from the Author.

Sorbonne Université

Laboratoire d'Informatique de Paris 6

UMR UPMC/CNRS 7606 – Tour 26

4 Place Jussieu – 75252 Paris Cedex 05 – France

Acknowledgments

I would like here to thank all the people that have helped me during this internship. Thank you to my tutor, Julien Tierny, aside from your awesome class to introduce TDA, your enthusiasm was contagious and your help precious.
In order of geometric proximity, thank you to Maxime Soler and Guillaume Favelier for sharing with me your unerring knowledge about the darkest C++ concepts and for always being ready to listen to or tell geeky jokes. Thank you to Charles Gueunet for allowing me to use his computer in order to run my performance tests, without your help, Chapter 4 would be much sparser. Thank you to Léa Sta for our deep debates on every topic imaginable from the meaning of life to the particularities of the British weather. Thank you to Solane El Hirch for sharing your unwavering good mood.
Thank you to Irphane Khan for taking care of all the paper work necessary for the internship.

Thank you to every other peprson working at the LIP6 that made any of my days brighter with a smile, a joke or a few nice words.

Finally, I would like to adress a special thank to Théo Lacombe who accepted to refactor his code to make it user-friendly (you did it great!) and sent it to me in order to compare performances. Thank you also for our fruitful discussion about specificities of your work.

Contents

List of Figures

Introduction

1

This manuscript is the result of a $5^{1/2}$ months research internship at the Laboratoire d'Informatique de Paris 6 (LIP6) under the supervision of Julien Tierny, it is also my Master Thesis as a part of the Master *Mathématiques, Vision, Apprentissage* (MVA) at the Ecole Normale Supérieure Paris Saclay (ENS). Its topic is *Wasserstein Barycenters of Persistence Diagrams*.

Topological Data Analysis (TDA) is an approach of the field of Applied Mathematics that aims at extracting topological information from datasets. It is particularly useful when confronted with high dimensional datasets where data tends to be sparse or if the data is corrupted by noise. It uses notions like *persistent homology* to give intuitions on the *shape* of the data being studied.
In particular, TDA uses Persistence Diagrams. Persistence Diagrams are $2D$ representations of the topology of a dataset. A natural distance to define over the space of Persistence Diagrams is the *Wasserstein distance* which is based on the concept of optimal transportation. With this distance, one can define the barycenter of a set of Persistence Diagrams as the Fréchet mean of this set of diagrams. Although this definition is simple, the computation of such barycenters is challenging. To our knowledge, two main algorithms solving this problem have been proposed. The goal of this Master Thesis is to introduce a new fast algorithm to compute the Barycenter of Persistence Diagrams and to study its performances.
The document is organised into 4 distinct chapters: Chapter 2 describes introduces the notions of Persistence Diagrams and their barycenters and describes the State of the Art on this topic, Chapter 3 introduces the algorithms developed during the internship, Chapter 4 is an experimental study of these algorithms in which comparisons with previous work are performed and implementation choices are justified, finally Chapter 5 introduces some ideas that seemed interesting in order to develop further

the algorithms of Chapter 3 and to improve their performances.

The scientific contributions of my internship are the following:

- Creation and implementation of the Partial Bidding Algorithm, an efficient algorithm to compute barycenters of Persistence Diagrams. This algorithm proves to be much faster than previous State of the Art algorithms on a CPU. In the conditions of our experiments we observed a gain of 3 orders of magnitude on a State of the Art algorithm.

- Creation and implementation of a progressive version of the Partial Bidding Algorithm that alows the user to define a time-limit to the computations and still get a precise approximation of the most persistent points of the barycenter in extremely rapid times.

- Inclusion of the Partial Bidding algorithm in the K-Means pipeline for unsupervised clustering and improvement of this setup by using further the paradigms of the Partial Bidding algorithm.

- Creation and implementation of a progressive clustering algorithm allowing the user to fix a time limit for its computations and to obtain precise clustering based on the most persistent points of the diagrams in extremely rapid times.

Presentation of the Problem 2

Contents

This chapter introduces the problem of the computation of barycenter of Persistence Diagrams with respect to the Wasserstein distance. It describes the State of the Art in this domain as well as some applications.

2.1 Persistence Diagrams and Wasserstein Distance

2.1.1 Persistence Diagrams

Recently, the field of topology has gained in importance in the domain of data analysis. In particular, the study of persistent homology became one of the keystones of Topological Data Analysis. Persistent homology tracks the appearance an disappearance of topological features at different scales in a dataset. The information obtained can be summarized in a *Persistence Diagram*.

Formally, a Persistence Diagram D is a set of points $\{(x, y) \in \mathbb{R}^2 | y \geq x\}$ with integer multiplicity which contains the diagonal $\Delta = \{(x, x) | x \in \mathbb{R}\}$ with infinite multiplicity. Each non-diagonal point carries information about a topological feature. We say that D is the *empty diagram* if $D = \Delta$.

Persistence Diagrams can be obtained from data through several processes. In this document, we will restrain our study to diagrams obtained from bidimensional scalar fields.
In order to understand the construction of Persistence Diagrams from scalar fields, let's introduce the *sub-level sets* of a function.
Let $f : \mathbb{R}^2 \to \mathbb{R}$ be a Morse function (a function twice differentiable with no degenerate point). We call the sub-level set at level $\alpha \in \mathbb{R}$ of f the set of all points in $\mathbb{R}^2$ such that their image is lower than or equal to α: $f_\alpha^- = f^{-1}(]-\infty, \alpha])$. Symmetrically, we define the sur-level sets of f as follows : $f_\alpha^+ = f^{-1}([\alpha, \infty[)$.
When α varies, so does the number of connected components of f_α^- and f_α^+. More specifically, let's define $CC_f^-(\alpha)$ and $CC_f^+(\alpha)$ these quantities. It is clear that CC_f^- and CC_f^+ are piecewise constant functions. Moreover, it can be shown that their discontinuities lie in the values of the critical points of f. Local minima increment CC_f^- as a new connected component appears in f_α^-, we say that a connected component is born ; at saddle points, connected components of f_α^- fuse and CC_f^- is decremented, we say that a connected component has died ; and by convention, we consider that at the global maximum, the last connected component of f_α^- dies. Symmetrical behaviours apply for sur-level sets.
Each time a connected component of sub-level sets dies at a value $\alpha = y$, we add to the Persistence Diagram of f a point (x, y) where x is the birth date of the youngest connected component of the connected components

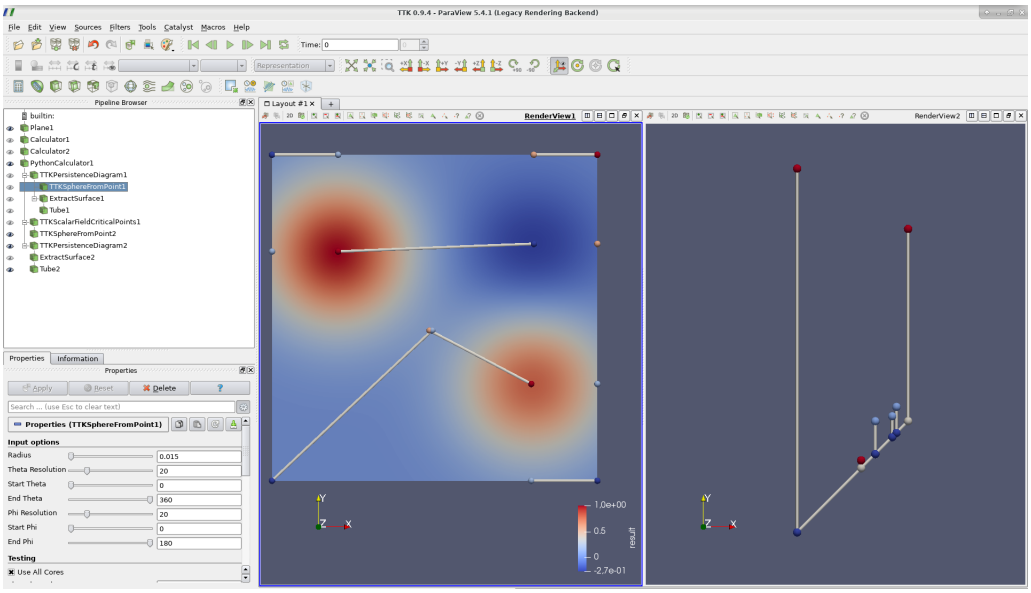

Figure 2.1 – *Example of a Diagram alongside the Scalar Field from which it was created (Screenshot from TTK/Paraview)*

fused at $\alpha = y$. Symmetrically, for sur-level sets, the point (y, x) is added when a connected component of sur-level sets is born at x and dies at y. Since a connected component can only die after it is born, this construction ensures that all points added to the diagram are above the diagonal. Moreover, each point in the Persistence Diagram corresponds to a pair of critical points of f: either a *local minimum - saddle point*, a *saddle point - local maximum* or a *global minimum - global maximum* pair.

Figure 2.1 shows an example of a Persistence Diagram obtained from the addition of 3 Gaussian. The scalar field is composed of 4 local minima (dark blue points) and 3 local maxima (red points). Each persistence pair in the Persistence Diagram is represented by a segment between two critical points in the original scalar field.

We call the *persistence* of a point $p = (x, y)$ in a Persistence Diagram the difference $y - x$, which corresponds to the lifespan of the topological feature summarized by p. Diagonal points therefore have a persistence equal to 0. The most persistent points correspond to large scales topological features. The addition of a low level of noise in a dataset can induce the presence of additional critical points. These critical points form persistence pairs, each corresponding to a point on the diagram. But since the noise level is low, the lifespan of the topological features generated by the critical pairs is low, and therefore, the points added in the persistence

diagram have a low persistence.

One of the reasons why Persistence Diagrams are used is their property of stability uder small perturbations: Persistence Diagrams are stable under small modifications of the scalar field they were generated from. This stability can be formally expressed by the notion of distance described in the next Section.

2.1.2 Wasserstein Distance, Matchings and Munkres Algorithm

One important feature of Persistence Diagrams is the possibility to compute distances between them using Optimal Transport based distances. Although the cardinal of two Persistence Diagrams is infinite, it is possible to transform this problem in a finite optimal matching problem between two sets of same cardinality and then use classical Optimal Transportation techniques.

Let D_1 and D_2 be two Persistence Diagrams. Let $\mathscr{G}$ be the set of bijective maps from D_1 to D_2 (points with multiplicity $q > 1$ are considered to be q different points). We call $m \in \mathscr{G}$ a matching between D_1 and D_2. Intuitively, a matching between D_1 and D_2 is pairing of each point of each diagram to another unique point of the other diagram.
We define the *cost* of a matching m between D_1 and D_2 as follows:
$C_{D_1,D_2}(m) = \sum_{p \in D_1} (1 - \mathbb{1}(p \notin \Delta)\mathbb{1}(m(p) \notin \Delta)) \, ||p - m(p)||_2^2$
Where $||.||_2$ is the Euclidean distance.
The cost of the matching is the sum of the costs of matching each pair of points $(p, m(p))$ where the cost of matching a pair is the Euclidean distance between the points of this pair if at least one of the points is off-diagonal. Otherwise the cost of the pair is 0.
Finally we call the 2-Wasserstein distance between D_1 and D_2 the minimal cost over all matchings:
$W_2(D_1, D_2)^2 = \min_{m \in \mathscr{G}} C_{D_1,D_2}(m)$

The problem of finding an optimal matching is highly combinatorial. However, Persistence Diagrams are infinite -and non-countable- sets of points. Therefore, the problem needs to be simplified to be computationnally doable.
Two essential remarks can be the base of the simplification of the problem:

- If a matching m is a minimizer of the cost, then, all non-diagonal

points paired up with a diagonal point are paired up with their orthogonal projection on the diagonal. (Otherwise, a less costly matching which verifies this property is easy to construct).

- Any number of pairs of diagonal points can be modified without changing the price of a matching.

Turner et al. (2012) and Kerber et al. (2016) suggest to use these remarks to reduce this infinite matching problem to a finite one.
Let D_1 and D_2 be two persistence diagrams. Let $D_1 = \{p_1, p_2, \dots, p_{n_1}\} \cup \Delta$ and $D_2 = \{q_1, q_2, \dots, q_{n_2}\} \cup \Delta$ where the p_i and q_i are off-diagonal points counted without their multiplicity.
For $p \in \mathbb{R}^2$, we define $\overline{p}$ to be the orthogonal projection of p on the diagonal Δ.
Finally, we define the *augmented Persistent Diagrams* $\overline{D_1}$ and $\overline{D_2}$ to be respectively $\{p_1, p_2, \dots, p_{n_1}, \overline{q_1}, \overline{q_2}, \dots, \overline{q_{n_2}}\}$ and $\{q_1, q_2, \dots, q_{n_2}, \overline{p_1}, \overline{p_2}, \dots, \overline{p_{n_1}}\}$.
The problem of the finding a matching of minimal cost between D_1 and D_2 can then be reduced to finding a matching of minimal cost between $\overline{D_1}$ and $\overline{D_2}$ which contain the same finite number of points. In order to extend an optimal matching m between the augmented diagrams to a matching of the same cost between the original diagrams, it is sufficient to append any matching between $\Delta - \overline{D_1}$ and $\Delta - \overline{D_2}$.
In the sequel, with a slight abuse a notation, we will call a *matching* between D_1 and D_2 a matching between their augmented diagrams.

The problem of the optimal matching in bipartite graphs (or optimal assignment) has been studied for a long period of time. One of the most classical algorithm to solve it is the Munkres (or Hungarian) algorithm (Munkres (1957)). Originally with a $\mathcal{O}(n^4)$ complexity this algorithm has been refined to a $\mathcal{O}(n^3)$ algorithm where n is the size of the diagrams to be matched.

Figure 2.2 shows an example of a matching between two Persistence Diagrams (viewed in 3D) obtained from the Munkres algorithm. Points are colored according to the type of critical point they were obtained from. (in this example, *min/saddle* pairs were matched to *min/saddle* pairs,and *saddle/max* or *min/max* were matched to *saddle/max* or *min/max*).
The pairing is represented by colored bars. The scale of colors goes from blue for low cost pairs to red for high cost pairs. Some points do not

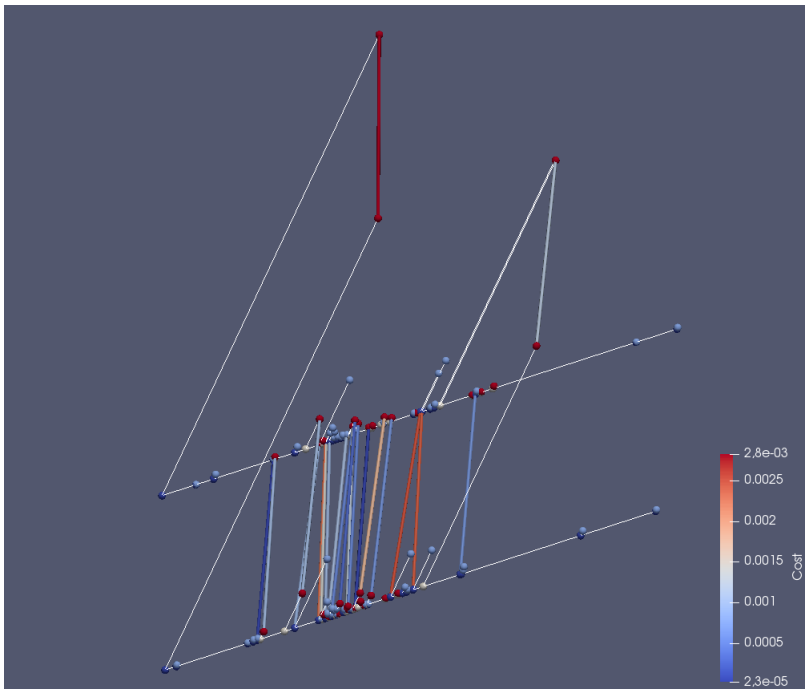

Figure 2.2 – *Example of a Matching between two Diagrams (Screenshot from TTK/Paraview)*

seem matched on the picture, this means that they were matched to their projection on the diagonal.

2.1.3 Barycenters and Motivation

Once the notion of distance between diagrams has been defined, the notion of barycenter can be generalized from the barycenter in Euclidean contexts. In $\mathbb{R}^n$, the barycenter of a set of points $(p_i)_{i\leq n}$ is generally defined as their arithmetic mean $\overline{p} = \frac{1}{n}\sum_{i\leq n} p_i$. However, an equivalent but more general definition can be given. Indeed the arithmetic mean verifies $\overline{p} = \underset{p\in\mathbb{R}^n}{\arg\min} \sum_{i\leq n} ||p - p_i||^2$.

Similarly to barycenter in Euclidean context, given a set of Persistence Diagrams $D_1, \ldots, D_n$, we define their barycenter (or *Fréchet Mean*) $\overline{D}$ as the Persistence Diagrams that minimized the sum of its squared distance to each diagram:
$\overline{D} = \underset{D}{\arg\min} \sum_{i\leq n} W_2(D, D_i)^2$.
Intuitively, this definition corresponds to a Persistence Diagram roughly "in the middle" of the D_i.
Note that the choice of the 2-Wasserstein distance and to square this distance is arbitrary in the definition, however, it is the choice that is the most

coherent with the idea of the barycenter in $\mathbb{R}^n$. Moreover, since the W_2 distance is closely related to the Euclidean distance, the methods exposed in this document extensively use the fact that finding the barycenter of Euclidean points is an easy problem whereas finding the minimizer of the sum of $\mathscr{L}^p$ distances can be a much harder problem if $p \neq 2$.

We show on Figure 2.3 an example of a barycenter between two diagrams. On the top row, we show the two Persistence Diagrams (D_0 and D_1 from left to right), the bottom right corner show the barycenter that was computed from these diagrams, and the bottom-left corner shows the matchings between each diagram and the barycenter (the original diagrams were overlaid on each other on the foreground, the color of each point represents the diagram it belongs to. The barycenter is on the background). The barycenter is represented without its diagonal for clarity purposes.
The low persistence pairs in D_0 and D_1 can be found back on the barycenter, and the common high-persistence pairs are also well-represented on the barycenter.
Note however that D_0 possesses one more high persistence pair than D_1. This additional pair translates into a smaller one in the barycenter. Since this barycenter point is far from its corresponding point in D_0, it is associatied with a high cost for this pair (dark red pairing), moreover, it is not paired up with any off-diagonal point from D_1.

The motivations behind the computation of barycenters of Persistence Diagrams are multiple.
Given several diagrams, one could need to summarize them in order to study them globally without being tricked into falsely generalizing the specificities of one particular Persistence Diagram of this set. Computing a barycenter is often the most natural choice in order to summarize a set of objects.
Moreover, some Machine Learning algorithms require the computation of barycenters. The K-Means algorithm, certainly the most widely used algorithm for unsupervised classification purposes, is an instance of such algorithms. The use of the K-Means algorithm on Persistence Diagrams has driven the research presented in this document. In Sections 3.4 and 4.2 we used the specificities of our algorithms to adapt the K-Means algorithm on the Persistence Diagram clustering problem in order to perform faster clustering.

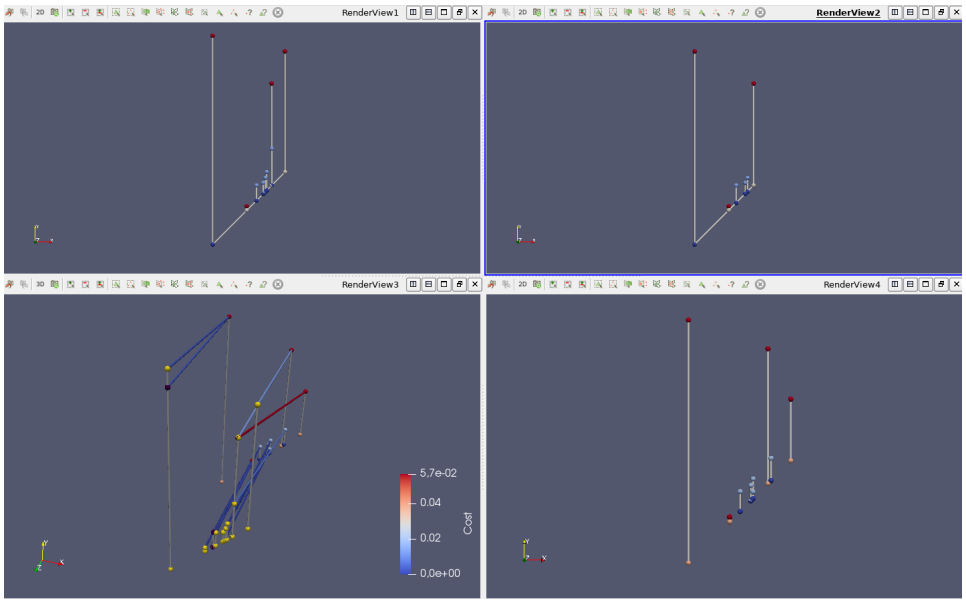

Figure 2.3 – *Example of a Barycenter between two Diagrams (Screenshot from TTK/Paraview)*

2.2 State of the Art

The goal of this Section is to present the State of the Art in matters close to Persistence Diagrams Barycenters.
We will first introduce some works about Reeb graphs which are extentions of Persistence Diagrams that take into account some geometrical information. It will raise some limits about Persistence Diagrams but will give a first look at the difficulties to overcome in order to be able to compute barycenters of Reeb graphs.
We will then suggest various approaches in order to compute barycenters of Persistence Diagrams.
Finally, we will describe in depth one of these approaches as well as a way to accelerate it via an approximate matching algorithm faster than the Munkres algorithm. This last part will describe the basis on which our work was built. We strongly encourage the reader to be at ease with the concepts it develops before reading Chapter 3.

2.2.1 Geometrical Extensions : Reeb Graphs

Persistence Diagrams carry information about the topology of the objects it describe, however, some objects can have similar topological information although being geometrically very different. For instance, Persistence Diagrams are invariant under rotation or translations of the objects they study. This property is often wanted when using Persistence Diagrams

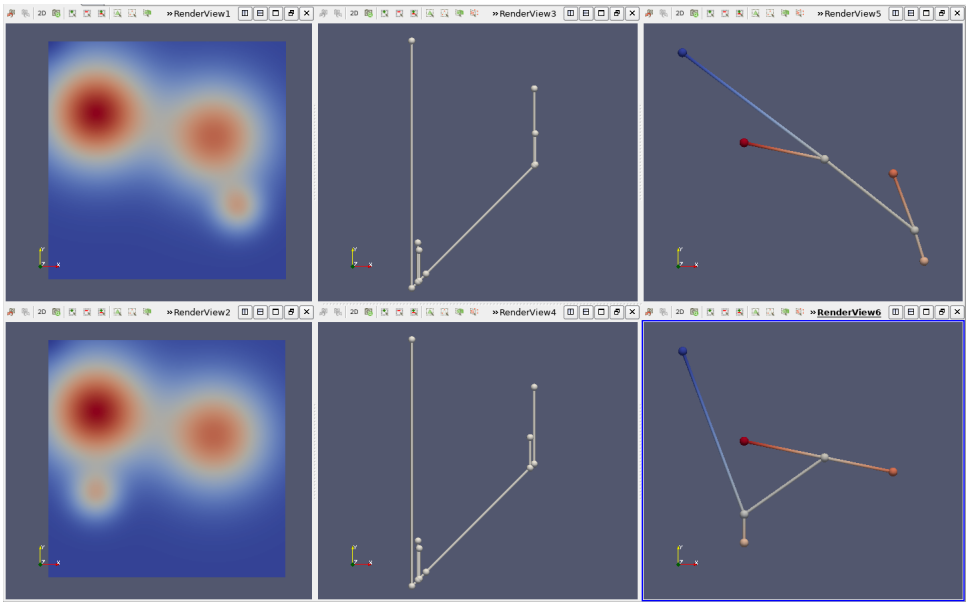

Figure 2.4 – *From left to right: Scalar Field, Persistence Diagram and Split Tree associated with it*

(for instance in the field of $3D$ object recognition, one does not want translations to affect the recognition of the object given). However, some other geometrical artifacts are important in the understanding of data. One of them is the nested structure of the topological features. Figure 2.4 shows two geometrically very different Scalar field giving birth to very similar Persistence Diagrams. Using only their Persistence Diagrams, it is impossible to differentiate these scalar fields, but in practical applications, being able to do may be crucial.

Reeb Graphs (a slightly more complex structure than the Split Tree, on the third column of Figure 2.4) summarize the topological features of scalar fields and keep track of their interdependence. Parsa (2013) has shown that they can be computed in $\mathcal{O}(m\ log(m))$ where m is the size of the 2-skeleton of the scalar field.
Formally, let's define a equivalence relation $\sim$ on the domain X of a scalar function f. We say that $x \sim y$ iff $f(x) = f(y) = \alpha$ and x and y belong to the same connected component of $f^{-1}(\alpha)$. Reeb (1946) defines R_f, the *Reeb Graph* of f to be equal to $X/\sim$. If f is a Morse function, R_f has a simplicial complex structure. The Reeb Graph can be used to understand the structure of the data.

Given two Reeb Graphs R_f and R_g Bauer et al. (2013) proposes a defintion of the distance between R_f and R_g:

- Let $\Phi : R_f \to R_g$ and $\Psi : R_g \to R_f$ be two continuous maps.
- Given u and v in the domain of definition of f, we define $d_f(u,v) = \min_{\pi : u \rightsquigarrow v} height_{R_f}(\pi)$, where π is a continuous path from u to v in R_f, and $height_{R_f}(\pi) = \max_{x,y \in R_f} (f(x) - f(y))$.
  d_f defines a distance between points in a Reeb Graph.
- Let $G(\Phi, \Psi) = \{(x, \Phi(x)) \,|x \in R_f\} \times \{(y, \Psi(y)) \,|y \in R_g\}$
- Let $D(\Phi, \Psi) = \sup_{(x,y),(\tilde{x},\tilde{y}) \in G(\Phi,\Psi)} \left(\frac{1}{2} |d_f(x,\tilde{x}) - d_g(y,\tilde{y})| \right)$
  $D(.,.)$ is a measure of how much Φ and Ψ distort R_f and R_g. If $R_f = R_g$ and $\Phi = \Psi = Id$, then, $D(\Phi, \Psi) = 0$.
- Finally, the *Functional Distortion* distance between R_f and R_g is defined as follows :

$$d_{FD}(R_f, R_g) = \inf_{\Phi,\Psi} [max \{D(\Phi, \Psi), ||f - g \circ \Phi||_\infty, ||g - f \circ \Psi||_\infty\}]$$

This definition is similar to the Gromov-Hausdorff distance but using continuous maps instead of isometries and this distance can be easily extended to merge trees or split trees.
Bauer et al. (2013) show that the space of Reeb Graphs equipped with the Functional Distortion distance is stable under small perturbations: $d_{FD}(R_f, R_g) \leq ||f - g||_\infty$.
The main problem of the Functional Distortion distance is that, to our knowledge, there does not exist any algorithm to compute it. (Bauer et al. 2013) believes that it could be computed in an exponential time, which is of course inefficient for large and noisy datasets which would have large Reeb Graphs. Moreover, this distance is similar to the $||.||_\infty$ norm, it seems to us improbable that computing a barycenter using it could be doable using methods similar to those of Section 2.2.3.

Similarly, Hilaga et al. (2001) defines a similarity between Reeb graphs, and Dey et al. (2015) defines a distance between Reeb Graphs using the bottleneck distance between Persistence Diagrams but it is not clear how to use them to compute efficiently a barycenter of Reeb graphs.

Reeb Graphs carry more information than Persistence Diagrams about how persistence pairs organize themselves, and in particular how different topological features are nested in one another. Distances defined on

Reeb Graphs take this information into account but are often using infinite norms that are unfriendly to classical gradient descent oriented algorithms and that are not adapted to the definition of barycenter (for instance, barycenter of Persistence Diagrams using the bottleneck distance are not unique). However, adding some penalty to matchings between Persistence Diagrams that do not respect the nestedness of the topological features could be a lead to extend our definition of barycenter of Persistence Diagrams. One drawback to this approach could be the presence of loops in the Reeb Gaphs (cf. Cole-McLaughlin et al. (2003)) that would limit the number of algorithms that could be used. Another one is the appearance of instabilities under slight perturbations, which loses one of the main interest of Persistence Diagrams.

2.2.2 Various Possible Approaches for the Computation of Barycenters

A first idea to avoid the problem of optimal transportation is to find a transformation of Persistence Diagrams into another space in which the distance resembles the Wasserstein distance but the computation of which does not require to solve an optimal transportation problem.
The transformation of Persistence Diagrams into *Persistence Images* introduced by Adams et al. (2017) achieves this goal.
Given a Persistence Diagram D and $u \in D$, we define:

- $f(u)$ the weight of u where f is piecewise differentiable and $f(\Delta) = \{0\}$. Adams suggests to choose f increasing with persistence and capped by 1 (for instance, $f(u) = max(1, persistence(u))$.
- $\phi_u(z)$ be a probability distribution centered around u. Adams suggests a Gaussian distribution and leaves the choice of the variance as an open problem.

The Persistence Image is a function $\rho : \mathbb{R}^2 \to \mathbb{R}$ such that $\rho(z) = \sum_{u \in D} f(u)\phi_u(z)$.
ρ defines a bidimensional scalar field wich smoothes the Persistence Diagrams: each Persisistence pair adds a Gaussian peak in the persistence image with a height depending of its persistence.
It was shown by Adams et al. that the Persistence Images are stable with respect to the Wasserstein distance, as for $k = 1, 2$ or ∞:
$||\rho(D) - \rho(D')||_k \leq \left(\sqrt{5}|\nabla f| + \sqrt{\frac{10}{\pi}}\frac{||f||_\infty}{\sigma}\right) W_1(D, D')$.
As a consequence, minimizing the difference between 1-Wasserstein distances ensures that Persistence Images are low. Unfortunately, the inverse

is not necessarily true.
This scalar field ρ can be discretized on a grid and then turned into a vector. Instead of directly computing the barycenter of a set of Persistence Diagrams, one can instead compute the barycenter of their Persistence Images in the Euclidean space they lie on. Adams et al. used this method to perform K-Means on several shapes. Not only the use of Persistence Images reduced the time needed for the classification (compared to the direct use of Persistence Diagrams and of Presistence Landscapes), but also improved the classification accuracy, showing that Persistence Images can yield positive results.
A drawback of this method is that there is no way to solve the inverse problem of Persistence Images: given a Persistence Image, there is no known algorithm to find the Persistence Diagram from which it was obtained. What's more, the barycenter of Persistence Image is not necessarily the Persistence Image of a Persistence Diagram.

Another way to avoid the combinatorial aspect of the Barycenter problem is to use a convex relaxation. This idea was introduced for distance between histograms in Cuturi (2013) where an entropic penalty was added to the Wasserstein distance, defining the *Sinkhorn distance* in which high entropy transportation plans are prefered to low entropy ones.
Given two histograms $r \in \mathbb{R}^d$ and $c \in \mathbb{R}^d$, and denoting $P \in \mathbb{R}_+^{d \times d}$ a transportation plan from r to c, the entropy of P is defined as
$h(P) = -\sum_{i,j=1}^{d} p_{i,j} log\ p_{i,j}$
Given a distance (or cost) matrix $M \in \mathbb{R}_+^{d \times d}$, the distance between r and c is $d_M(r,c) = \min_P \langle P|M \rangle$.
A high entropy transportation plan tends to distribute each bin in r uniformely in every bin of c. On the opposite a low entropy and low cost transportation plan will tend to distribute each bin of r in the closest bins of c.
The Sinkhorn distance of parameter λ is defined as follows:
$d_M(r,c) = \langle P^\lambda | M \rangle$ where $P^\lambda = \min_P \langle P|M \rangle - \frac{1}{\lambda} h(P)$
The optimal transportation plan P^λ can be found using the iterative Synkhorn-Knopp's fixed point algorithm.
The parameter λ is however hard to tune: the Synkhorn distance converges to the classical optimal transportation distance when λ tends towards infinity. On the other hand, lower values of λ make the algorithm faster to converge. There is therefore a balance to find between speed of

execution and precision of the result.
The Sinkhorn distance was later used to compute barycenters of probability measures in Cuturi and Doucet (2014). The algorithm alternates between computing optimal transportation plans between every probability measure and the current candidate to be barycenter and updating the positions of the barycenter weights at fixed transportation plan. The Sinkhorn algorithm intervenes in both of these steps where the transportation plan and the positions of the barycenter weigths are computed using an entropic penalty.
Finally, Lacombe et al. (2018), extended these techniques to the computation of Barycenter of Persistence Diagrams. To do so, they encoded their Persistence Diagrams as histograms on a regular square grid and computed the barycenter on the same grid using gradient descent and entropic penalization.
The algorithm implemented has several advantages on previously existing methods: not only is it faster to achieve convergence, but it is also focused on GPU implementation. This was possible thanks to the fact that the Sinkhorn algorithm runs matrix operations that are independant diagramwise. The GPU allows to parallelize the computations for up to thousands of diagrams, making the algorithm scalable.
The main drawback comes from the entropy penalization which tends to spread points into a seemingly continuous heatmap visually similar to a Persistence Image which is not a Persistence Diagram. The penalization factor λ tends to change drastically the output of the algorithm. Low values of λ tend to create mass near the diagonal. This makes it hard to choose the parameters of the algorithm and to interprete its outputs.

We saw that using Persistence Images to compute barycenters requires to change the space of inputs into a Euclidean space and the outputs lie in the same Euclidean space. On the other hand, the algorithm described in Lacombe et al. (2018) keep the inputs in the space of Persistence Diagrams but outputs a heatmap which is not a Persistence Diagram. The later utilizes Wasserstein distances and, in this sense, solves a problem of Wasserstein Barycenter contrarily to the first one which is mainly heuristic since it does not use the Wasserstein distance nor does it give any theoretical guarantee on the Wasserstein distance achieved.

2.2.3 Expectation Maximization and Auction Bidding

In this section, we introduce two crucial algorithms that we will use in the sequel. The first one is an algorithm introduced by Turner et al. (2012) which computes a Barycenter in the space of Persistence Diagrams.

The second part will present the Auction algorithm used to compute an approximation of the optimal matching between two sets of points of same cardinality. It was first introduced by Bertsekas (1981) and later completed by Kerber et al. (2016) in order to compute distance between Persistence Diagrams.

We will use these two algorithms together in Chapter 3 in order to solve the Barycenter problem on a CPU in seconds, even for very large diagrams.

Expectation Maximization Algorithm

We introduce in this Section the algorithm described in Turner et al. (2012), for the sake of clarity, due to its structure, we took the liberty to call it the Expectation-Maximization Barycenter algorithm (EMB). Throughout this Section, we will denote $D_1, \dots, D_n$ the diagrams from which we want to compute the barycenter.

The idea behind this algorithm is similar to the one used in the Lloyd algorithm, it keeps a candidate barycenter B throughout the computations, and alternatively refines the matchings between the D_i and B and the position of the points of B.
The EMB algorithm is intialized with a candidate barycenter B (in practice, we chose one of the D_i at random). The goal is to iteratively refine B to lower the sum of its distances to the datasets D_i. Each step of the algorithm will lower the sum of these distances. Therefore, the candidate barycenter tends towards a local minimum in a gradient-descent fashion. In the sequel, with a slight abuse of language, we will call this candidate barycenter the *current barycenter* or the *barycenter*.

1. The first step is to find an optimal matching ϕ_i between each of the datasets D_i and the candidate barycenter B. Turner and al. suggest to use the Munkres algorithm to do so.
   After this step, for all $i \leq n$, each point $b^{(j)} \in B$ is matched with a

unique point $d_i^{(j)} \in D_i$[1].
This is the *Expectation* (or *Matching*) step of the algorithm.

2. The second step consists in updating the position of $b^{(j)}$ to minimize the sum of the Wasserstein distances at constant matchings $(\phi_i : Di \to B)_{i \leq n}$. This is, by definition of the 2-Wasserstein distance, equivalent to defining $b^{(j)} \leftarrow \arg\min_{b \in \mathbf{R}^2} \sum_{i=1}^{n} ||b - d_i^{(j)}||_2^2$.
This problem has a well known solution, which is the average of the $d_i^{(j)} : b^{(j)} = \frac{1}{n} \sum_{i=1}^{n} d_i^{(j)}$.
Moreover, some of the $d_i^{(k)}$ may be matched with a diagonal point $b \in \Delta$ of B (which is therefore necessarily the projection of $d_i^{(k)}$ on Δ). In this case, a new point $b_{new}^{(i,k)}$ is added to the barycenter. This point is chosen to minimize the cost of its matching to $d_i^{(k)}$ in diagram D_i and to the diagonal in all other diagrams. Therefore $b_{new}^{(i,k)} = \frac{1}{n} d_i^{(k)} + \frac{n-1}{n} Proj_\Delta(d_i^{(k)})$.
This is the *Maximization* step (or *Barycenter update*).

3. Once the barycenter has been updated, the algorithm goes back to Step 1. The termination criterion is met if two consecutive sets of matchings $(\phi_i : Di \to B)_{i \leq n}$ are exactly the same for all $i \leq n$, This choice is justified by the fact that, if this condition is met, a local minimum has been found and the barycenter would not change anymore if the algorithm kept iterating.

The process is summarized in Algorithm 1. The pseudo code of the *UpdateBarycenter* function is described in 2.

It can easily be shown that both the computation of the matchings and the update of the barycenter make the cost of the Barycenter decrease. Therefore, since the cost of the Barycenter is positive, the number of possible matchings is finite and since a given matching generates a unique position for the points of the barycenter, it can be concluded that the EMB algorithm converges.

Contrarily to the algorithms exposed in the previous section, the output of the EMB is a Persistence Diagram itself. However, computing it requires numerous calls to the Munkres algorithm (n per Expectaion step). This fact makes the algorithm unpractical for diagrams with more than a dozen of points.

[1] which is either a point off-diagonal or the projection of $b^{(j)}$ on the diagonal of D_i according to Section 2.1.2

Algorithm 1: Compute the Fréchet mean B of Persistence Diagrams D_1, ... D_n

Input : Diagrams D_1, ..., D_n

Output : Barycenter B

$B = D_i$ One of the Diagrams chosen at random
$finished \leftarrow False$

while not $finished$ **do**
  for i in 1..n **do**
    // Find the optimal matching between D_i and B
    $\Phi_i := (d_i^{(j)}, b^{(j)}) \leftarrow Munkres(D_i, B)$
  end for
  $B \leftarrow UpdateBarycenter(\Phi_1, \ldots, \Phi_n)$
  if no Matching Φ_i has changed **then**
    $finished \leftarrow True$
  end if
end while
return B

Algorithm 2: UpdateBarycenter : Updates the barycenter

Input : Matchings $\Phi_i = (d_i^{(j)}, b^{(j)})_j$ for all $i \leq n$, current barycenter B

Output : Updated Barycenter B

for $b^{(j)} \in B$ **do**
  $b^{(j)} \leftarrow \frac{1}{n} \sum_{i=1}^{n} d_i^{(j)}$
end for
$B \leftarrow (b^{(j)})$

for i in $1 \ldots n$ **do**
  for $d_i^{(k)}$ in D_i **do**
    if $d_i^{(k)}$ is matched with a diagonal point of B **then**
      Append $b = \frac{1}{n} d_i^{(k)} + \frac{n-1}{n} Proj_{\Delta}(d_i^{(k)})$ to B
    end if
  end for
end for
return B

This limitation is the key motivation for us to consider the introduction of the Auction Bidding algorithm.

Auction Bidding

In this section, we describe the Auction Bidding algorithm, an iterative algorithm that computes an aproximation of an optimal matching given two sets with identical number of points and a matrix of matching costs for each pair of points that do not belong to the same set. This algorithm was first introduced by Bertsekas (1981). We will describe the algorithm and try to provide intuitive insights as to why it works, readers wishing to know its theoretical foundations are encouraged to read the article aforementionned. Note that the hypothesis stating that the two sets must contain the same number of points is not restrictive in our context of Persistence Diagrams since, as we described it in Section 2.1.2, we do not compute matchings between diagrams directly but between augmented Persistence Diagrams, that, by construction, contain the same number of points.

Let D and B be two Persistence Diagrams augmented with each other (D contains the projections on Δ of the off-diagonal points of B and vice-versa).
Let $d_1, \dots, d_q$ and $b_1, \dots, b_q$ be the points of B and D respectively. The Auction Bidding algorithm replicates the functionning of real life auctions in which points of B would *buy* the points of D. We will therefore call the d_i *bidders* and the b_i *objects* [2]. We will say that a bidder *owns* an object if it is matched with this object, the bidder will then be called the *owner* of the object.
Each bidder d extracts some *benefit* from each object b equal to the opposite of the cost of the pairing (b, d):

$$ben_d(b) = \begin{cases} -||b-d||_2^2 & \text{if } b \text{ and } d \text{ are both off-diagonal} \\ -||b-d||_2^2 & \text{if } b \text{ is off-diagonal and } d \text{ is its projection on } \Delta \\ -||b-d||_2^2 & \text{if } d \text{ is off-diagonal and } b \text{ is its projection on } \Delta \\ 0 & \text{if } b \text{ and } d \text{ are both on the diagonal} \\ -\infty & \text{otherwise} \end{cases}$$

[2] The use of the letter b to denote objects and d to denote bidders may seem unfortunate, but this choice was made since in the computation of barycenters, bidders will be points of diagrams D_i and objects will be points of the barycenter B.

The definition of the benefit was chosen in order to have, given a matching m, the following equality $C_{D,B}(m) = -\sum_{d \in b} ben_d(m(d))$.
The price for an off-diagonal point and a diagonal point which is not its projection on the diagonal is set to $-\infty$ because in an optimal matching, no point can be paired to a diagonal point which is not its projection. This particular case is not *stricto sensu* needed in the definition of the benefit, but using such a distinction helps the algorithm to converge more easily.

Each object b is assigned a *price* $p(b)$ (initially $p(b) = 0$). For a bidder d, we define the *value* of an object b as the diffence between the benefit it extracts from this object minus its price: $val_d(b) = ben_d(b) - p(b)$.

The algorithm iteratively constructs the matching, during its execution, we will say that bidders matched to an object are *assigned* and bidders that are not matched will be said to be *unassigned*. Each bidder tries to maximize the value it gets from the object it is matched with. Intuitively, this means that bidders are assigned to objects close to them and with low prices.
Initially all bidders are set to be unassigned, all objects are assigned a 0 price. A parameter $\epsilon > 0$ is fixed.

The algorithm can be described as follows:
While there are some unasssigned bidders, let d be one of them. Let b be the object of maximal value for d. d is assigned to b, if b already had an owner, this previous owner gets unassigned. Let b' be the object with second highest value for d. Let's define a *price increment* $\Delta p = val_d(b) - val_d(b') + \epsilon$. The price of b is then increased by Δp.

The choice of Δp can be justified as follows:
If $\epsilon = 0$, Δp is the smallest increase of $p(b)$ that ensures that if another bidder bids on b, b will not be the most valuable object to d anymore. A smaller increase could lead to situations in which a new bidder d' bids on b, immediately followed by d bidding again on b, which would have added two useless operations for the same matching.
ϵ is set to be strictly positive for convergence purposes. If two bidders give the exact same value to their two most valuable objects, they would keep stealing each other's property without increasing its price. Intuitively, a

choice of a large ϵ discourages bidders to bid on objects owned by another bidder because their price gets artificially high as soon as the object has been bid on. In the limit in which $\epsilon = \infty$, once bidders are assigned to an object, they would never be unassigned since the price of this object would be infinite and therefore its value would be $-\infty$.
Obviously, discouraging bidders to bid on objects they would benefit from is not a good strategy in order to get a low cost matching. In particular, one can show that ϵ is directly linked with the cost of the final matching outputted by the Auction Bidding algorithm. If we denote C the cost of the matching outputted by the Auction Bidding algorithm, and $\hat{C}$ the cost of the optimal matching, Bertsekas proved the following theoretical bound on C:

$$C \leq \hat{C} + q\epsilon$$

The lower the value of ϵ, the more precise the matching is. However, in practice, low values of ϵ also make the algorithm slower to converge. Bertsekas suggests to use *ϵ-scaling* to workaround this difficulty: start by computing the matching for large values of ϵ, then start again after dividing ϵ by 5 with prices initialized with the prices of the previous Auction.
Iterating this strategy refines the matching at each iteration. The initialization of the prices at their previous value makes each iteration relatively fast. Bertsekas suggests to use an initial value for ϵ equal to a quarter of the largest cost possible for a pair of points matched. It can be proved that no point can be matched at a cost larger than twice the cost of matching it to its projection on Δ. Therefore, using $\epsilon = \frac{1}{2} \max_{p \in B \cup D} persistence(p)^2$ seems to be a recommended initialization.
Kerber et al. (2016) states that iterated enough times, this procedure would lead to the optimal matching. However, this would take too many iterations. instead of that, it is possible to choose a relative margin of error $\delta > 0$ and stop whenever the cost of the matching found C verifies $\delta \leq \frac{C}{C - n\epsilon} - 1$. Thanks to the inequality on costs shown by Bertsekas, Kerber et al. (2016) proved that, under this condition, $C \leq (1 + \delta)\hat{C}$.

The Auction Bidding algorithm is summarized in Algorithm 3

For the sake of clarity, in the sequel, we will call the whole matching procedure *Auction*, one iteration of the global auction (the call to the *AuctionRound* function) will be called an *Auction Round* and the assignement of a bidder to an object and the corresponding price change (line 6 of

Algorithm 3: **Matching** : Compute an approximation of the optimal matching between augmented Persistence Diagrams B and D

Input : Augmented Persistence Diagrams B and D, with q points each, δ maximum accepted relative precision on the matching cost

Output : δ-approximation of the optimal matching

$\epsilon \leftarrow \frac{5}{2} \max_{p \in B \cup D} persistence(p)^2$ (max distance between points)
$C \leftarrow 0$
while $\delta \leq \frac{C}{C - n\epsilon} - 1$ **do**
  $\epsilon \leftarrow \epsilon / 5$
  $\Phi \leftarrow AuctionRound(B, D, \epsilon)$
  $C \leftarrow C_{B,D}(\Phi)$
end while
return Φ

Algorithm 4), a *Bidding*.

Algorithm 4: **AuctionRound** : Performs an Auction Round

Input : Augmented Persistence Diagrams B and D with q points each, $\epsilon > 0$

Output : Approximation of the optimal matching

while There exists a bidder $d_j \in D$ unassigned **do**
  Find object b_k with best value and object b_l with second best value
  if a bidder d_m is assigned to b_k **then**
    Unassign d_m
  end if
  Assign d_j to b_k
  $p(b_k) \leftarrow p(b_k) + val_{d_j}(b_k) - val_{d_j}(b_l) + \epsilon$
end while
return Matching

The most costly part of the Auction algorithm is the search for the two objects with highest value. The naive way to do it is to perform an exhaustive search on all objects. Since the value of an off-diagonal object for an off-diagonal bidder is the opposite of the sum of its distance to the bidder and of its price ($val_d(b) = -||b - d||_2^2 - p(b)$), at equal prices for all objects, the highest value objects for d are its neighbouring objects. A logical improvement to retrieve them would therefore be to consider some geometrical information. For this reason, Kerber et al. (2016) intro-

duced the use of K-D Trees to perform the search of the most valuable off-diagonal objects.

K-D Trees can be used to perform efficient searches for k nearest neighbours (for any $k \in \mathbb{N}$). Each node of the tree contains one object, and each subtree contains all objects in a certain bouding box. The K-D Tree has the structure of a binary tree. Each node divides a bounding box (the whole space for the root of the tree) into two non-overlapping bounding boxes containing the same amount of points. Knowing the bounding box associated with each subtree allows the nearest neighbours searches to avoid certain parts of the trees where all points lie farther than the k^{th} closest neighbour already found.
In order to use K-D Trees to find the 2 most valuable off-diagonal objects for a given bidder, we need to augment them with some price information. Kerber suggests to add the price of each object in its corresponding node as well as the information of the lowest price in the subtree below this node. With this structure, the algorithm for the search of the 2 most valuable off-diagonal objects will look in a subtree if and only if the bounding box it defines is at a distance lower than the value of the second most valuable object already found minus the minimal price in the subtree.

An off-diagonal bidder can be paired up with any off-diagonal point or with its projection on Δ, therefore, in order to find the 2 most valuable objects for a given bidder d, one can find its 2 most valuable off-diagonal objects using this augmented K-D Tree, and compare them with its projection on Δ.
On the opposite, diagonal bidders can only be paired up with another diagonal point or with an object they are the projection of on Δ. Since the value of a diagonal object for a diagonal bidder is the opposite of its price, geometry plays no role in the matching of a diagonal bidder and a diagonal object. Kerber suggests to use *lazy heaps* to store diagonal objects order by increasing price.
Lazy heaps are similar to priority queues except that when a value needs to be updated, it is updated only when it is on first position of the heap. If the price of a diagonal object is modified during the Auction algorithm, the lazy heap does not change. However, when the lazy heap is asked for its lowest price element, it first checks if price of its first element is up to date. If it is, the lazy heap returns it, otherwise, the price of the first element is updated, and the element positionned back in the correspond-

ing position in the heap. This behaviour guarantees to always return the lowest price object when asked for it (which is what a priority queue is made for), and minimizes the number of updates necessary to keep the queue up to date [3].
In order to find the two most valuable objects of a diagonal bidder d, the algorithm is therefore the following: get the first two objects in the lazy heap. While their prices are not up to date, update them and reorder the heap. If both their prices are up to date, compare their values to the value of the only off-diagonal object which projection is d[4]. Keep only the two most valuable.

Kerber used the Auction algorithm in order to compute the distance between Persistence Diagrams faster. Section 4.1.1 shows that using K-D Trees is indeed faster the original algorithm of Bertsekas, which is, itself, faster than the Munkres algorithm.

Our first contribution was to use the Auction algorithm for the computation of the barycenter of Persistence Diagrams. The *Expectation* step of the EMB algorithm requires the computation of matchings between each of the diagrams D_i and the current barycenter B. Instead of computing these matchings via the Munkres algorithm, we did it using the Auction algorithm. Some more improvements to the EMB algorithm were made by taking advantage of the structure of the EMB algorithm and of the Auction algorithm. It is the main topic of Chapter 3.

2.3 Applications

In this section, we will briefly introduce some example in which barycenters of Persistence Diagrams can prove useful.

[3] Without the queue's *laziness*, each time the price of an off-diagonal objects would be updated (for instance when an off-diagonal bidder would bid on it), the corresponding object would have to be found in the priority queue, then its price updates, and then, the queue would have to be reordered. This could be implemented in $\mathcal{O}(log\ q)$ with q the number of diagonal objects
The Lazy heap still needs to update the position of some objects, but it does it less often since the price of an object could change several times before it gets on the first position in the heap

[4] If there are several of them, they created several overlaid diagonal bidder, each of them associated with only one of these objects

One of the first applications historically tackled by Persistence Diagrams is the recognition and classification of $3D$ shapes. Computing the Persistence Diagram of a three-dimensional shape requires the definition of a scalar field on the surface of the shape, then, the pipeline described previously can be followed. For more on this topic, the reader can refer to Edelsbrunner and Harer (2010).
Lacombe et al. (2018) for instance performs unsupervised clustering on a $3D$-shape dataset contaning faces, heads, elephants, horses, camels and cats. Munch et al. (2015) uses barycenter to compute statistics on point clouds from which Persistence Diagrams were extracted.

Computing barycenters of any representation of an ensemble of datasets can be a way to summarize it. This makes sense in particular when the datasets are similar.
In the field of Physics simulation, a slight change of parameters modifies the ouput of a simulation, but, in general, its global structure is not drastically changed. In Gunther et al. (2014), the results of multiple simulations (for instance, in fluid mechanics, simulations of vortices represented by the magnitude of the velocity field for different values of viscosity) with slight changes in the choice of parameters are summarized in a histogram field: each point of the domain of definition contains a histogram summarizing all values taken by the scalar field on this point. This representation is then used to compute areas of mandatory critical points: closed areas were, given the empirical distribution[5] of values taken in previous simulations, there must be at least 1 critical point.
Mandatory critical points are a way to summarize an ensemble of scalar fields. Computing the Barycenter of Persistence Diagrams can be another way to summarize it. Using localization of critical points in the Persistence Diagrams (cf. Section 3.1.4) could make the Barycenter and the mandatory critical point techniques complementary, each area of mandatory critical point could be linked to a point in the Barycenter corresponding to persistence pairs found in that area.

Ferstl et al. (2015) and Ferstl et al. (2016) aim at visualizing and classifiying streamlines and isocontours. Using a Euclidean representation and dimension reduction (via PCA), a K-Means clustering is easy to compute. Moreover, an inverse transformation from the PCA-space to the original

[5]In practice, only the minimum and maximum value at each point are used.

data domain space exists. Confidence ellipses in the PCA space can be converted into a variability plot in the domain space. Doing so makes it possible to draw centroids as well as a confidence domain around it in the domain space.
However, these approches discriminate steamlines based on geometrical features. It makes it difficult to discriminate between geometrically close steamlines but displaying different topological features.
Combining these approaches with topological ones could improve the clustering by discriminating geometrically different steamlines as well as topological different ones. Still, finding a visual representation of the topological differences is not an easy problem. Using the mandatory critical points approach is a solution: the mandatory critical points area can be represented cluster-wise in the domain space like in Favelier et al. (2018).

3 The Partial Bidding Algorithm

Contents

This Chapter is the core of this manuscript. It describes the contributions of my internship:

- The creation of a new algorithm called *Partial Bidding Algorithm* for the computation of barycenters of Persistence Diagrams.

- Its extension to the use of the K-Means algorithm which, if done properly, can reduce the complexity of the clustering.
- How to compute the barycenter and the K-Means progressively, allowing the user to define time limits for its computation and to get satisfing results for the most persistent points.
- The parallelization of the Expectation Maximization algorithm.

3.1 Partial Bidding

In this Section, we introduce the *Partial Bidding Algorithm*. It is based on the Expectation-Maximization approach described in Section 2.2.3 combined with the Auction Bidding algorithm.

We will denote B the barycenter at each step of the algorithm and $D_1, \ldots, D_n$ the diagrams from which B is computed.

3.1.1 Global Idea: Approximations can be Approximated

The Expectation-Maximization algorithm iteratively refines the barycenter using alternatively two computational bricks:

1. Given the current barycenter B, compute the optimal matching between B and each of the D_i
2. Given the matching previously found, update B to minimize its cost.

This process refines B and is ensured, if the optimal matching is found exactly, to lower the price at each step. The output is not necessarily the barycenter of the D_i, but is a local minimum of the cost function of the barycenter. Moreover, at each step of the algorithm B is an approximation of this local minimum that gets more and more precise in a gradient descent-like approach.
It was observed in the field of optimization, that using an approximation of the gradient instead of the real gradient still lead to good results. The *Stochastic Gradient Descent* used in Machine Learning for instance uses this fact to accelerate convergence by considering only a portion of data collected at each gradient computation. The *Stochastic Gradient Descent* uses the fact that each descent of gradient refines an approximation of the local minimum that the algorithm will return and states that approximating this approximation at each step is not problematic as long as the convergence of the result is not affected too much.

Using the Auction Bidding algorithm to compute the optimal matching during the Matching step of the EMB algorithm follows the same logic: the Matching step is performed with finite precision. This leads to an approximation of the barycenter that would be obtained after using the Hungarian algorithm during this Matching step, namely an approximation of an approximation.

Although the Matching step of the algorithm can increase the cost of the barycenter, the maximal increase is controlled by the δ parameter. This ensures that, in the worst case, the cost of the barycenter increases at most of δ at each iteration. If δ is small enough, this quantity is neglictable, and, in practice, the cost of the barycenter generally keeps decreasing.
Our first suggestion is to use the Auction Bidding algorithm instead of the Munkres algorithm during the Matching step of the EMB algorithm. In practice, the EMB algorithm using the Auction Bidding converges to similar results than with the Hungarian algorithm in similar number of iterations but each of these iterations is made much faster by the Auction Bidding algorithm.

3.1.2 Memorizing Previous Results

We stated in Section 2.2.3 that the first Auction Rounds of the Auction Bidding algorithm were used to obtain a convergence of the prices of the points of B. In the worst of cases prices are increased by ϵ at each bidding, which, if ϵ is too small can cause a slow convergence of prices.
In the Expectation Maximization algorithm used with the Auction Bidding, after each Matching step, all information about how this matching was found is lost. In particular, the final prices of points of B are not saved in memory. However, we expect B not to move a lot between two successive iterations, particularly in the late stages of the algorithm when the convergence is almost reached. If B does not move a lot, the benefit each bidder of D_i gets from an object of B stays similar[1] and therefore, the prices bidders are ready to pay do not change a lot.
This last remark justifies the fact that initializing the prices of objects to their final prices during the previous Matching Step seems a better choice than arbitrarily fixing them to 0 at the start of the Auction Bidding algorithm. This strategy avoids costly first iterations of the Auction Bidding algorithm.
Note that before the matching step, each point of B must be matched with a point in each of the D_i, otherwise, it would have been deleted during the barycenter update step. This means that for each point of B, n prices

[1] For a bidder d and an object b, initially separated by a square distance $D^2_{d,b}$, if B is updated to B' such that $W_2(B, B')^2 = \Delta^2$, then b was translated at most of a distance Δ to a point b' and therefore $|D^2_{d,b} - D^2_{d,b'}| \leq D_{d,b} \times \Delta + \Delta^2$.
This affirmation is therefore particularly relevent for bidder-object couples that are close together, which is generally the case for couples that were matched together in the Matching Step.

must be saved in memory after each Matching Step. We denote $p_i(b)$ the price of the object b in the matching between B and D_i.
One case remains to be discussed: matching bidders to the diagonal results in the addition of new points in B for which initial prices must be determined. Without any previous price reference for them, we chose to initialize their prices with the average price of the previous objects of B in its matchings with the D_i: $p_i(b_{new}) = \frac{1}{|B_{old}|} \sum_{b \in B_{old}} p_i(b)$.

We implemented this strategy and compared it to the basic EMB algorithm using Auction without memory on Section 4.1.2. Figure 4.2 shows that this optimization alone does not improve the computational time of the algorithm (See curves $(1) + (2)$ and (1)).
This is due to the fact that a clever initialization of the prices makes the first Auction Rounds useless but does very little to change their complexity. The Partial Bidding algorithm (Section 3.1.3) will get rid of these first Auction Rounds and make more radical approximations in order to gain computational time.

3.1.3 Partial Bidding, Laziness Helps

The goal in this section is to introduce our solution to the barycenter problem: the **Partial Bidding algorithm**. It achieves two goals: suppressing the need for the computation of the early Auction Rounds in the Matching step of the barycenter computation and accelerating the Matching step by making it more approximate without losing precision in the final output.
These two goals can be respectively associated with not using the early Auction Rounds of the Auction Bidding algorithm during which prices rapidly converge (which is useless if the prices are already precise enough thanks to the price memorization described in the previous Section) and not using the latest Auction Rounds during which the matching gains in unnecessary precision. We therefore used a single Auction Round between consecutive barycenter updates. The initial prices for this round are determined by the final prices of the previous Auction Round. The Partial Bidding Algorithm is therefore identical to the EMB algorithm with Auction Bidding on all points except the fact that it performs a single Auction Round instead of waiting for the convergence of the Auction Bidding algorithm at each matching step.
Only the ϵ parameter needs to be specified in order to adjust the precision of the matching during this Auction Round.

Recall that low values of ϵ tend to make the Auction Round longer since it takes more biddings for the prices to get to their optimal value, but it also makes it more precise. Since we want our algorithm to be fast, ϵ needs to be sufficiently important, but in order to obtain a convergence towards a local minimum in the latest iterations of the algorithm, we need ϵ to be decreasing.

How to choose ϵ? In order to answer this question, let's consider an example. Let B be the current barycenter of $D_1, \dots D_n$ during the Partial Bidding algorithm. Suppose that at the Update step, only one point b of B is moved to $b + \delta b$. Let $d \in D_1$ be the bidder associated with b.
We have : $||b + \delta b - d||^2 = ||b - d||^2 + 2\langle b - d|\delta b\rangle + ||\delta b||^2$.
Therefore, the benefit of b for d is decreased by $2\langle b - d|\delta b\rangle + ||\delta b||^2$, and therefore the price that d is ready to pay for b is decreased by the same quantity.
If the translation δb was chosen in a random direction uniformely distributed over $[0, 2\pi[$, the expectancy of $2\langle b - d|\delta b\rangle$ would be 0. Under these conditions, the expected absolute modification of the price that d is ready to pay for b is $||\delta b||^2$.
Moreover, in the worst case (if two objects have the same value for a bidder), during an Auction Round, prices are increased by ϵ at each bidding. If $\epsilon \ll ||\delta b||^2$, then it will take on average at least $\frac{||\delta b||^2}{\epsilon} \gg 1$ biddings for the price of b converge towards its new optimal value. Therefore, it seems natural to choose $\epsilon \geq ||\delta b||^2$. However, ϵ is the minimal price change when a bidding occurs, therefore, if $\epsilon \gg ||\delta b||^2$, the value modification that occured when the barycenter was updated gets neglictable compared to the price modifications induced by the biddings which results in a loss of precision. Therefore, for the sake of precision, it also seems natural that $\epsilon \leq ||\delta b||^2$.
These remarks empirically justify the heuristic of $\epsilon = ||\delta b||^2$ in this toy example.

In the general case, we generalized this choice to $\epsilon = \max_{b \in B} ||\delta b||^2$ but this choice led to two major issues :

1. If the matching did not change during two successive iterations, the barycenter would not change either during the Update step, leading to $\epsilon = 0$ and, as a consequence, possibly an infinite loop during the following Auction Round.

2. Although we experimentally observed that such a choice of ϵ tended to make the barycenter converge and therefore to make ϵ tend towards 0, nothing guarantees this convergence. Some cases of non-convergence of ϵ and therefore, of B are, as a consequence, possible.

We solved the first issue by forcing ϵ to decrease at most exponentially at each iteration: $\epsilon \leftarrow max(\frac{\epsilon}{5}\ ,\ \max_{b \in B} ||\delta b||^2)$. In the case of a constant matching, this would be equivalent to performing several consecutive Auction Rounds without any barycenter update with ϵ decreasing at the same rate as suggested by Bertsekas in the Auction Bidding algorithm until a change in the matching occurs.
The convergence of ϵ towards zero can simply be forced by using a decreasing upper bound on its value.
Finally, we implemented the following choice for the update of ϵ:

$$\epsilon \leftarrow \max\left(\frac{\epsilon_0}{n^2_{iterations}}\ ,\ max\left(\frac{\epsilon}{5}\ ,\ \max_{b \in B} ||\delta b||^2\right)\right)$$

We called this choice of ϵ the *ϵ-decrease* optimization. For the sake of comparison, in Figure 4.2, we compared this choice of ϵ (Curve $(1)+(2)+(3)$) with a constant value of $\epsilon = 2.10^{-5}$ (Curve $(2)+(3)$) and observed that it could be up to 100 times faster.

Algorithm 5 synthesizes the Partial Bididng Algorithm. Note that the K-D Tree does not require to be computed inside the loop line 11 since it can carry the information of the n prices per object in a unique structure.

Figure 3.1 shows the result of the Partial Bidding algorithm applied to find the barycenter of 4 Persistence Diagrams. The figure shows from left to right and top to bottom:

1. One of the original 2D scalar-field used. They all were computed by adding some Gaussian noise to a two-dimensional sine, then were smoothed using a Gaussian filter.

2. The Persistence Diagram obtained from this scalar-field.

3. The matchings between the 4 Persistence Diagrams (showed overlaid on the foreground) and their barycenter. The color represents the cost of matching each pair.

4. The barycenter.

Algorithm 5: Partial Bidding Algorithm

Input : Persistence Diagrams D_1, ..., D_n

Output : Barycenter B

$B = D_i$ One of the Diagrams chosen at random
$finished \leftarrow False$

$\epsilon_0 \leftarrow \frac{1}{2} \max_{p \in D_i,\ i \leq n} persistence(p)^2$
$\epsilon \leftarrow \epsilon_0$
$P_i \leftarrow (0)_{b \in B}$ for $i \leq n$

while not $finished$ **do**
  $kdt \leftarrow$ K-D Tree of B with prices included.
  for i in 1..n **do**
    $\Phi_i := (d_i^{(j)}, b^{(j)}) \leftarrow AuctionRound(D_i, B, \epsilon, P_i, kdt)$
    $P_i \leftarrow$ final prices of the objects in the Auction Round
  end for
  $B \leftarrow UpdateBarycenter(\Phi_1, \ldots, \Phi_n)$
  $\epsilon \leftarrow \max\left(\frac{\epsilon_0}{n^2_{iterations}},\ max\left(\frac{\epsilon}{5},\ \max_{b \in B} ||\delta b||^2\right)\right)$
  For every P_i, delete prices of points of B deleted during the update.
  Add prices in all the P_i for points added in B during the update.

  if ϵ is small enough **or** cost of B has increased twice in a row **then**
    $finished \leftarrow True$
  end if
end while
return B

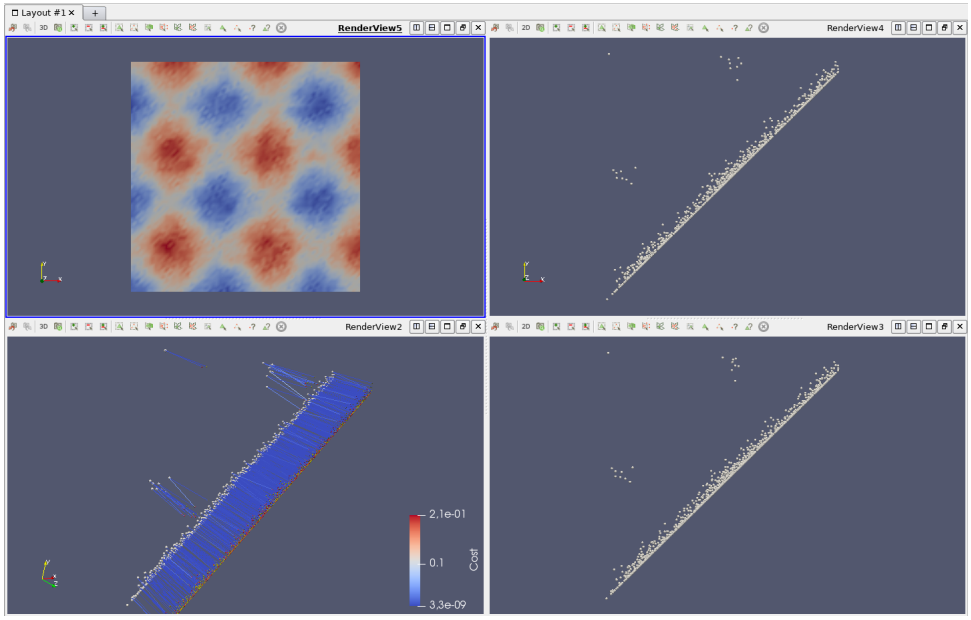

Figure 3.1 – *Example of a barycenter of* 4 *diagrams. From left to right and top to bottom: one of the original 2D scalar-field, its Persistence Diagram, the matchings between the* 4 *(overlaid) Diagram and their barycenter, the Barycenter (Screenshot from TTK/Paraview)*

Each of the 4 Persistence Diagrams used contained between 567 and 594 off-diagonal points. The barycenter was computed in 3 seconds. It contains 4 main groups of points corresponding to 4 groups of points in each of the input Persistence Diagrams: one point of high persistence isolated, 2 groups of 8 points each of medium persistence and numerous low persistence points due to noise.

3.1.4 Extension to 5D diagrams

Our Partial Bidding algorithm can easily be enriched to include geometrical information. Instead of using a Persistence Diagram as a collection of points lying in $\mathbb{R}^2$, one can add some dimensions corresponding to the positions of the critical points giving birth to a persistence pair in the space of data as suggested by Soler et al. (2018a)eperi.
For an initial dataset in $\mathbb{R}^d$, two choices can be made:

- Work with a persistence diagram as a collection of points in $\mathbb{R}^{2+2d}$, in this setup the first two dimensions correspond to the original Persistence Diagram, the $2d$ next dimensions correspond to the coordinates of the critical points of the dataset that generated the critical pair in the Persistence Diagram
- Or work in $\mathbb{R}^{2+d}$ with 2 dimensions for the classical Persistence Diagram and the d others storing the coordinates of the middle of the

segment (in the data domain) between the pairs of critical points generating the critical pairs of the Persistence Diagram.

We implemented the second of these options. The K-D Tree structure is particularly adapted to this extension since it can manage arbitrary number of dimensions. The results showed matchings between points of the Persistence Diagrams that were more geometrically consistent. This can be seen as an intermediate solution between barycenter of Persistence Diagrams and barycenter of Reeb Graphs mentioned in Section 2.2.1 and is an effort to include geometrical considerations into the computation of barycenters. The Partial Bidding algorithm in a $2 + d$-dimensional setup only differs from Algorithm 5 in the computation of distances between bidders and objects that are now performed in $\mathbb{R}^{2+d}$.

3.1.5 Assignment Order and Barycenter Initialization

The Partial Bidding Algorithm implicitely makes two non-trivial choices that have been left undiscussed up to this point: how to initialize the barycenter B and in which order do unassigned bidders bid? We explored several possibilities to answer each of these questions. In this Section, we present these possibilities and, as a part of the negative results obtained during this internship, explain that they did not improve our performances.

The problem of the barycenter initialization is the only non-deterministic part of the Partial Bidding algorithm. Different choices for the initial barycenter can yied different results. In particular, some choices may lead to poor local minima. (Lacombe et al. 2018) showed that even for very simple sets of Persistence Diagrams, there may not always exist an initialization leading the EMB algorithm to converge towards the global optimum.
We tried several approaches to tackle this issue. Initializing B to the empty diagram or a concatenation of all input Persistence Diagrams is not a good idea since the first iteration of Partial Bidding will update B by projecting each point of all D_i towards the diagonal (with a persistence $\frac{1}{n}$ times their original persistence) and in practice, leads to poor results both in terms of execution time and final barycenter's cost. Choices like the largest or the smallest Persistence Diagram gave similar performances in terms of final barycenter cost than the initial random choice of barycenter. Therefore, we chose to keep choosing the initial barycenter randomly among the D_i.

This strategy has an advantage over the deterministic ones: it allows to run the algorithm several times with different initializations and opt, as a result for the barycenter with minimal cost.

At the level of the Auction Round, unassigned bidders are ordered in a queue. The first bidder of the queue bids and is deleted from it. Bidders that get unassigned are added at the back of the queue. Initially, all bidders are put in the queue. There is no obvious reason why their initial order in that queue should not have any influence on the complexity of the algorithm. In particular, if an object is very disputed between several bidders, it may be optimal to order their biddings so that the price of the object rises to its final value in a minimal number of operations[2].
In order to follow this heuristic, we tried ordering the bidders by price paid during the previous Auction Round, by the time of their last assignment in the previous Auction Round, by persistence, by benefit or value obtained in the previous Auction Round, but none of these orders had a significant impact on the time of computation.

These negative results led us to believe these two topics are not decisive for the performance of the algorithm. We encourage the reader seeking potential improvement for the Partial Bidding algorithm to read Chapter 5 in which some ideas of optimizations are suggested.

3.2 Progressive Barycenter

In this Section, we introduce our second main contribution: the progressive computation of the barycenter. This approach has the ambition to be user-friendly. Its goal is that the barycenter B carries useful information at all time during the Partial Bidding algorithm so that the user can define a time limit for the computation of the barycenter, and get, after this time-limit a barycenter (with a low number of points if the time-limit was low) that has converged and therefore that can be used for practical purposes. To do so, the algorithm will focus on priority on the most persistent points of the Persistence Diagrams since they are the one carrying the principal

[2]In practice, this would require knowing in advance which bidder has the largest difference of value between this object and its second most valuable object. This is an expensive information to compute. However, some heuristics may help. For instance, a bidder that paid a high price in the previous Auction Round is more likely to be linked to a very disputed object.

information.
It allows a visual evolution of the algorithm and open new possibilities for online problems in which time limits are strict. Moreover, as Section 4.1.3 will show, this paradigm also improved the execution time of the Partial Bidding algorithm.

3.2.1 Specificities of Persistence Diagrams and Goals of Progressive Barycenter Computation

Persistence Diagrams are more than point clouds. Each of their points carries topological information corresponding to a topological feature of some data. Some features have more importance than other: for example, when measuring a temperature at each point of a domain, measurement errors can be the cause of tiny peaks in the scalar field obtained. These peaks contain a local maximum (or local minimum) that create a persistence pair in the Persistence Diagram. However, due to their noisy nature, these peaks are not relevent in the analysis of the data. This is relected by the fact that the corresponding persistence pair has a low persistence since the noisy peaks are tiny. On the contrary, high persistence pairs tend to carry the intrinsic topological structure of the data. Low persistence pairs tend to carry information insignificant or noisy topological features.

Computing barycenters of Persistence Diagram is costly. In order to shorten the execution time of the algorithm used, it is tempting to simplify the diagrams beforehand by keeping only the most persistent pairs because they contain the most important information on the data. This way, the barycenter is faster to compute and summarizes the principal information contained in the diagrams.
However, in some cases, the user may want to obtain the most complete barycenter possible in a given amount of time (during live analysis of some phenomenons for example). Choosing a persistence threshold for the simplification of the input data that respects a time limit and does not simplify too much the diagrams is far from being trivial in this case. Our Progressive Barycenter approach is adapted for this kind of usage. It automatically fixes a persistence threshold under which no point is taken into account in the input Persistence Diagrams, and outputs the corresponding barycenter within the time limit.

3.2.2 Adding Points to Refine the Barycenter

This Section introduces the Progressive Barycenter algorithm. The idea behind the progressive computation of the barycenter is to progressively add points to the input Persistence Diagrams during the Partial Bidding process. The algorithm keeps a persistence threshold ρ in memory and to decrease it as the value of ϵ in the Partial Bidding algorithm diminishes.

Let $\rho \in \mathbb{R}^+$ be the *persistence threshold*.
For all $i < n$, we note $D_i^\rho \subset D_i$ the subdiagram of D_i containing only the diagonal and points of D_i with a persistence higher than ρ.

Our method is the following :

1. $\rho = \rho_0$ is initialized at a well chosen value. We decided to choose $\rho_0 = \frac{1}{2} \max_{i \leq n} (\text{max persistence of } D_i)$. Without loss of generality, we suppose that B is initialized with $D_0^{\rho_0}$.
2. A round of auction is performed with the D_i^ρ, and B and ϵ are updated as described previously in Algorithm 5.
3. ρ is decreased and the diagrams are enriched with points of persistence higher than ρ.
4. The barycenter is enriched with all points of D_0 with persistence between the previous value of ρ and its current value. The prices of these new barycenter points are fixed to the average price of all the previous points of B.
5. Iterate over from Step 2.

The main detail to tackle is how to decrease ρ when ϵ decreases. Let's suppose that, at a given iteration of the progressive algorithm, B is the exact barycenter of $D_1^\rho, \ldots, D_n^\rho$. We also assume that the points of B will not be affected by the addition of lower persistence points in the input diagrams when the persistence threshold diminishes[3].
Under these hypotheses, when new bidders with persistence between ρ and $\rho' < \rho$ are added to the input diagrams, the algorithm should treat them as if they were independant from the previously used bidders and therefore run *as if* it was computing the barycenter of

[3]These assumptions represent the perfect case for our algorithm, when working with real data, they are rarely met.

$D_1^{\rho'} \setminus D_1^{\rho}, \dots, D_n^{\rho'} \setminus D_n^{\rho}$. The initial value of ϵ for this computation is then determined by $\epsilon = \frac{1}{2} \max_{p \in D_i^{\rho'} \setminus D_i^{\rho},\ i \leq n} persistence(p)^2$.
For the sake of simplicity, we suggest to approximate this value by $\epsilon = \frac{1}{2}\rho^2$.

We extended this remark to the general case in which the aforementionned assumptions do not hold and defined the function $\phi : \rho \to \frac{1}{2}\rho^2$ and its inverse, $\phi^{-1} : \epsilon \to \sqrt{2\epsilon}$.
Whenever $\phi^{-1}(\epsilon) < \rho$, we updated $\rho' \leftarrow \phi^{-1}(\epsilon)$, enriched the diagrams with bidders of persistence between ρ and ρ', fixed $\epsilon \leftarrow \phi(\rho)$ and $\rho \leftarrow \rho'$.
The Progressive Barycenter algorithm is described in Algorithm 6.

In practice, in order to avoid brutal increases of the number of bidders, which would cause the iteration execution time to increase rapidly and make the user's time limit (that will be introduced in the next Section) less efficient, we controlled the number of bidders added in the diagrams when the persistence threshold is decreased. At line 21 of the Progressive Barycenter algorithm, we added a restriction on the decrease of ρ and chose ρ' such that $Card(D_i^{\rho'} \setminus D_i^{\rho}) \leq 10 + \frac{1}{10} Card(D_i^{\rho})$.
This control ensures that, if ϵ decreases abruptly from one iteration to another or if many bidders have similar persistence, the algorithm does not add too many new bidders to the diagrams which would go against the goal of progressivity of the algorithm.

The progressive algorithm takes a different path than the Partial Bidding algorithm in the space of Persistence Diagrams to converge towards a local minimum. For this reason, we expect it to converge towards a different local minimum. Section 4.1.3 shows the results of experiments comparing the performance of the Patial Bidding algorithm and of the Progressive algorithm. The Progressive algorithm tends to output poorer barycenters (Figure 4.4), but is also faster to do so (Figure 4.3).

3.2.3 User-Defined Time Limit

One of the objectives of the progressive computation of the barycenter was to allow the user to stop the computation whenever needed. This can already be done in the Partial Bidding setup but is far less efficient: since all iterations are managed with all bidders from the D_i, each Auction Round is costly during the Auction Bidding. On the opposite, the first

Algorithm 6: Progressive Barycenter Algorithm

Input : Persistence Diagrams D_1, ..., D_n

Output : Barycenter B

$\rho \leftarrow \frac{1}{2} \max_{i \leq n}$ (max persistence of D_i)
$B = D_i^\rho$ One of the Diagrams chosen at random
$finished \leftarrow False$

$\epsilon_0 \leftarrow \frac{1}{2} \max_{p \in D_i,\ i \leq n} persistence(p)^2$
$\epsilon \leftarrow \epsilon_0$
$P_i \leftarrow (0)_{b \in B}$ for $i \leq n$

while not $finished$ **do**
  $kdt \leftarrow$ K-D Tree of B with prices included.
  for i in 1..n **do**
    $\Phi_i := (d_i^{(j)}, b^{(j)}) \leftarrow AuctionRound(D_i^\rho, B, \epsilon, P_i, kdt)$
    $P_i \leftarrow$ final prices of the objects in the Auction Round
  end for
  $B \leftarrow UpdateBarycenter(\Phi_1, \ldots, \Phi_n)$
  $\epsilon \leftarrow \max\left(\frac{\epsilon_0}{n_{iterations}^2},\ max\left(\frac{\epsilon}{5},\ \max_{b \in B} ||\delta b||^2\right)\right)$
  For every P_i, delete prices of points of B deleted during the update.
  Add prices in all the P_i for points added in B during the update.
  if $\epsilon < \phi(\rho)$ **then**
    // Update the persistence threshold
    $\rho' \leftarrow \phi^{-1}(\epsilon)$
    $\epsilon = \phi(\rho)$
    $\rho \leftarrow \rho'$
  end if

  if $D_i == D_i^\rho$ for $i \leq n$ **then**
    if ϵ is small enough **or** cost of B has increased twice in a row **then**
      $finished \leftarrow True$
    end if
  end if
end while
return B

Auction Rounds performed in the Progressive Barycenter setup are fast since few bidders are considered. Therefore, satisfying results can be obtained from the Progressive Barycenter algorithm even after a very short time of execution.

We observed experimentally that, when the addition of new bidders was stopped after a given time t but the execution was continued until convergence (resulting in a barycenter computed on $D_1^\rho, \dots, D_n^\rho$ where ρ is the persistence threshold at time t), after $10t$ almost all barycenters have converged and around half of them converged after $3t$.
We used this observation to allow the user to choose a time limit t_{max} for the exectution of the algorithm. The Progressive algorithm is executed until $\frac{1}{10}t_{max}$. At this stage, the persistence threshold is fixed and no bidder is added to the diagrams anymore. The algorithm is left to converge and eventually stopped when t_{max} is reached if the convergence was not obtained before [4]. The output of the algorithm is therefore a barycenter computed on subdiagrams of the intial Persistence Diagrams. Since diagrams are enriched progressively, high values of t_{max} results in richer barycenters. Figure 4.5 on Section 4.1.3 shows the evolution of the number of bidders taken into consideration as t_{max} increases.

3.3 Parallelism

The nature of the Expectation-Maximization Barycenter and of the Partial Bidding algorithm makes them highly parallelizable. The Matching step is, by far, more costly than the Barycenter Update step, therefore, we focused our parallelization efforts on it.
Two options are available: either parallelize the Auction Round function, bidders would then be bidding simultaneously ; or parallelize the matching of each D_i to B. For the sake of simplicity, we chose to work on this second solution. Section 5.1 introduces some ideas for the parallelization of the Auction Rounds.

We used parallel computation at line 10 of Algorithm 5, when computing the matchings of each D_i with B. These matchings are independant from each other, therefore, their realization in parallel is trivial. However, it has some serious drawbacks.

[4] For the sake of clarity, we did not include the possibility of choosing a time limit in the pseudo-code of the Progressive barycenter described in Algorithm 6.

Let T be the number of threads available and n the number of Persistence Diagrams in input. The most favourable case is the case in which there are n threads available and each of the matching of D_i and B requires the same amount of operations. In this case, the n matchings end at the exact same time t at each iteration of the Partial Bidding instead of nt for the non-parallelized algorihtm, the speed-up is therefore, by definition, equal to n.
With $T = n$, in general, the execution time will be determined by the longest matchings to compute at each iteration of the Barycenter algorithm.
If $T < n$ the best theoretical speed-up possible is T, but as soon as some matchings are more costly than others to compute, the speed-up will drop. In particular, the order in which the matchings are computed matters. If one of them is a lot more costly than the other, it is more efficient to compute it first. We leave the problem of finding which matchings are likely to be more costly to compute open. Our implementation uses the natural order of computation (first, match D_1 and B, then D_2 and B, etc.). We studied the effects of parallelization on the Partial Bididng algorithm and show the results in terms of speed-up in Section 4.1.4.

3.4 Beyond Barycenters: the K-Means Algorithm

Until this Section, we focused on computing barycenter of Persistence Diagrams with two main objectives in mind: creating summaries of datasets or using them for clustering purpose via the K-Means algorithm.
In this Section, we will explore how to use these barycenter in the K-Means algorithm and how to extend the methods introduced by the Partial Bidding algorithm in order to compute more efficiently the K-Means clustering than by using the basic K-Means algorithm.

For the purposes of this Section, we will consider n Persistence Diagrams $D_1, \ldots, D_n$ to be classified in $1 < k \leq n$ clusters. For $j \leq k$, we will call C_j the j^{th} cluster, namely the set of Persistence Diagrams belonging to the j^{th} cluster, and B_j the centroid of this cluster.
The goal of the clustering is to minimize the following cost function:

$$\mathscr{L}_C = \sum_{j \leq k} \sum_{D \in C_j} d(B_j, D)^2$$

In our case, the distance function d is the Wasserstein distance W_2.

3.4.1 Basic and Accelerated K-Means

The K-Means algorithm is certainly the simplest and the best-known unsupervised classification algorithm. It is based on the alternative realization of two basic steps, each of them decreases the cost of the clustering:

1. Initially, the barycenters B_j are chosen randomly among the D_i.
2. Each Persistence Diagram D_i is assigned to the cluster C_j corresponding to its closest centroid B_j.
3. Centroids are updated to be the barycenters of the diagrams assigned to their clusters.
4. While the algorithm has not converged, it goes back to Step 2.

The algorithm stops when two consecutive iterations outputted the same clustering: at this point a local minimum is obtained and there is no point in continuing the iterations since the centroids and the clustering would not change anymore.
In the context of Persistence Diagrams, Step 2 can easily be done thanks to the Auction Bidding algorithm. It requires the computation of $n \times k$ distances between diagrams. And step 3 can be done using the Partial Bidding algorithm.
However, both these steps can be improved. In this Section, we present the Accelerated K-Mean algorithm introduced by Elkan (2003) which goal is to make the assignment to clusters (Step 2) less costly. Section 3.4.2 brings the Partial Bidding paradigm one step further to accelerate the computation of the centroids (Step 3).

The key observation in the use of the Accelerated K-Means algorithm is that the triangle inequality holds under the 2-Wasserstein distance.
thanks to this fact, the accelerated K-Means strategy observes that, it is not to compute all distances when looking for the closest centroid of a given diagram. Indeed, given a diagram D and two centroids B_0 and B_1 we have:
$W_2(B_0, B_1) \leq W_2(B_0, D) + W_2(D, B_1)$.
Therefore : $W_2(B_0, B_1) - W_2(B_0, D) \leq W_2(D, B_1)$.
If $W_2(B_0, B_1) > 2W_2(B_0, D)$, then, we necessarily have:
$W_2(B_0, D) \leq W_2(D, B_1)$.
The knowledge of $W_2(B_0, D)$ and $W_2(B_0, B_1)$ can therefore be sufficient to know that B_1 is not the closest centroid to D.

The strategy of the accelerated K-Means is therefore the following. Before assigning the input Persistence Diagrams to their closest cluster, the Accelerated K-Means algorithm requires to compute the matrix of distances between centroids $M = \left(W_2(B_i, B_j)\right)_{i,j \leq k}$. For a diagram D_i, let c_i be the cluster it was assigned to during the previous iteration of the K-Means. The search for the new closest centroid starts with the comparison of $W_2(B_{c_i}, D_i)$ to each distance $W_2(B_{c_i}, B_j)$ for all $j \neq c_i$. The distance $W_2(B_j, D_i)$ is computed if and ony if $W_2(B_{c_i}, B_j) \leq 2W_2(B_{c_i}, D_i)$. Otherwise, the inequality $W_2(B_j c_i, D) \leq W_2(D, B_j)$ would hold and B_j would not be strictly closer to D_i than B_{c_i}.
In the best of cases, this strategy needs the computation of $\mathcal{O}\left(k^2 + n\right)$ distances between diagrams[5] instead of n^2 for the classical update of the clustering.

Elkan (2003) also suggests to save in memory upper and lower bounds on distances between the D_i and the centroids in order of the number of distances to compute. This strategy is, once again, based on the triangle inequality:

$$W_2(B, D) - W_2(B, B + \delta B) \leq W_2(D, B + \delta B) \leq W_2(B, D) + W_2(B, B + \delta B)$$

where $B + \delta B$ is the position of B after being updated.

When centroids are updated, the distance of which they are translated is added to all upper bounds and substracted to all lower bounds. Each time a distance between an input Persistence Diagram D and a centroid B is required, the accelerated K-Means checks if lower or upper bounds are sufficient to conclude if B is or not the closest centroid to D. If it is the case, these upper and lower bounds on $W_2(B, D)$ are used instead of the real distance, and therefore, save a computation of distances. In the opposite case, the new distance is computed and the corresponding upper and lower bounds are updated. We chose not to go into too much detail about the accelerated K-Means, we invite the reader to read Elkan's article for a detail implementation.
In the best case scenario, with these upper and lower bounds, finding the closest centroid to a Persistence Diagram is as expensive as verifying a single inequality. In practice, if the clusters are well separated, this best case scenario is almost always achieved and the accelerated K-Means is

[5] In detail, $\frac{k(k-1)}{2}$ computations for the matrix of distance between centroids, and n to compute the new distance between each diagram and their previous centroid.

extremely fast.
Note that the Accelerated K-Means is a strict improvement of the K-Means algorithm since its complexity is reduced without any loss of accuracy on the final clustering.

We implemented these optimizations in our algorithm. The results of our experiments are shown on Figure 4.9 and 4.10 of Section 4.2.2 In practice the accelerated K-Means performed 2 to 4 times faster than the non-accelerated algorithm.

In order to have a faster convergence and to avoid getting stuck in local minima, we decided to use the K-Means++ initialization for our clustering algorithm. This strategy is described along with other initialization strategies in Celebi et al. (2012).

3.4.2 K-Means and Partial Bidding

In the previous section, we described how to accelerate the search for the closest cluster of each Diagram using the Accelerated K-Means algorithm introduced by Elkan. In the section, we focus on the other half of the algorithm: the update of the position of the centroids. The most direct approach would be to compute each centroid using the Partial Bidding algorithm at each iteration. This method would provide a precise centroid for each cluster at each iteration of the K-Means algorithm. However, during early iterations, similarly to the computation of the gradient in the Stochastic Gradient Descent algorithm or to the computation of the intermediate barycenter during the Partial Bidding algorithm, the clustering does not need to be perfect but only approximate (otherwise, a single iteration of the K-Means algorithm would always be sufficient, which is rarely the case in reality), therefore, the centroids are only approximations of the final centroids that will be found by the algorithm. Computing overly precise centroids in early stages of the K-Means algorithm therefore goes against the paradigm that led our reflexion about the Partial Bidding algorithm: *Approximations can be approximated*. Moreover, during two consecutive iterations, the clustering is rarely totally modified. As a consequence, centroids will not change substantially between two updates. Based on this fact, it seems natural to use information about the previous position of centroids before computing the new ones. We suggest to take the Partial Bidding algorithm one step further in order

to follow this direction and we introduce the Partial Bidding K-Means algorithm.

In short, the Partial Bidding K-Means algorithm performs a single Auction Round per iteration and per cluster instead of a full computation of the barycenter at each step of centroid update. The points of the centroids are objects which have prices with respect to each Persistence Diagram assigned to their cluster. These prices are kept in memory and are used to initialize the succeding Auction Round. ϵ decreases according to how much the centroids move between iterations.
If a Diagram D is assigned to a new cluster, its new centroid C has no prices corresponding to its previous matching to D. We therefore initialize them to 0. However, with a poor initialization of prices, performing an Auction Round with a low value of ϵ is inefficient. In order to make the prices converge to better values, we decided to run the Auction Bidding algorithm between D and C until a good enough precision on the prices is obtained[6], and only then, to perform the same Auction Round on D than on the other input Persistence Diagrams.
We chose to decrease ϵ in a similar way than in the Partial Bidding algorithm: ϵ is updated after the centroid update phase to the maximal translation of any point in a centroid.

Algorithm 7 describes in more depth the Partial Bidding K-Means algorithm. Note that in this form, the Partial Bidding algorithm for the computation of barycenters is a particular case of the Partial Bidding K-Means algorithm with $k = 1$ cluster.

We tested this algorithm and show the results in Sections 4.2.1 and 4.2. Our experiments proved that the Partial Bidding K-Means algorithm performs better than the direct application of Partial Bidding on the K-Means algorithm.

3.4.3 Progressive K-Means

Finally, we introduce our ideas for the progressivity of the barycenter computation into the K-Means setup. Using the same equivalence between the ϵ parameter and the persistence threshold ρ: $\epsilon \leftrightarrow \frac{1}{2}\rho^2$ as in the

[6]In practice, when the ϵ parameter of the Auction Bidding gets lower than the ϵ parameter of the Partial Bidding K-Means algorithm.

Algorithm 7: Partial Bidding K-Means algorithm

Input : Persistence Diagrams D_1, ..., D_n, k number of clusters

Output : Classification of D_1, ..., D_n into k clusters

for $j \in \{1, \dots, k\}$ **do**
  Initialize B_j to one of the D_i using the K-Means++ initialization.
end for
$finished \leftarrow False$
$\epsilon_0 \leftarrow \frac{1}{2} \max_{p \in D_i,\ i \leq n} persistence(p)^2$
$\epsilon \leftarrow \epsilon_0$
while not $finished$ **do**
  $(C_i)_{i \leq n}$, $M \leftarrow UpdateClusters(D_1, \dots, D_n, B_1, \dots, B_k)$
  for $j \leq k$ **do**
    $kdt \leftarrow$ K-D Tree of B_j with prices
    for i such that $C_i = j$ **do**
      $\Phi_i, P_i \leftarrow AuctionRound(D_i, B_j, P_i, kdt)$
    end for
    $(B_j) \leftarrow UpdateBarycenter((\Phi_i)_{C_i = j})$
  end for
  $\epsilon \leftarrow \max \left(\frac{\epsilon_0}{n_{iterations}^2} , max \left(\frac{\epsilon}{5} , \max_{b \in B_j,\ j \leq k} ||\delta b||^2 \right) \right)$
  For every P_i, delete prices of points of B deleted during the update.
  Add prices in all the P_i for points added in B during the update.
  Update lower and upper bounds on distances between D_is and B_js using the triangle inequality
  if ϵ is small enough **or** the cost of the clustering has increased twice in a row **then**
    $finished \leftarrow True$
  end if
end while
return Clustering $(C_i)_{i \leq n}$

Algorithm 8: UpdateClusters

Input : Persistence Diagrams $D_1, \ldots, D_n$, current centroids $B_1, \ldots, B_k$

Output : Current classification of D_1, ..., D_n into k clusters, and distance matrix between centroids.

for $j < m \leq k$ **do**
  $M_{j,m} \leftarrow AuctionBidding(B_j, B_m, \delta = 0.01)$ // Compute the distance between B_j and B_m with a precision of 1%.
end for
for $i \leq n$ **do**
  $C_i \leftarrow \arg\min_{j \leq k} W_2(D_i, B_j)$ // Find closest centroid using the Accelerated K-Means tricks.
end for
return $(C_i)_{i \leq n}$, M

Algorithm 9: UpdateCentroids

Input : Persistence Diagrams $D_1, \ldots, D_n$, current centroids $B_1, \ldots, B_k$, Current prices for centroids' objects $P_1, \ldots, P_n$

Output : Current classification of D_1, ..., D_n into k clusters, and distance matrix between centroids.

for $j \leq k$ **do**
  $kdt \leftarrow$ K-D Tree of B_j with prices included.
  for i such that $C_i = j$ **do**
    if D_i in new to cluster C_i **then**
      $P_i \leftarrow (0)_{b \in B_j}$
      $AuctionBidding(D_i, B_j, \epsilon_{min} = \epsilon)$
      $P_i \leftarrow$ Prices found during the Auction Bidding
    end if
    $\Phi_i := (d_i^{(l)}, b_j^{(l)}) \leftarrow AuctionRound(D_i, B_j, \epsilon, P_i, kdt)$
    $P_i \leftarrow$ final prices of the objects in the Auction Round
  end for
  $B_j \leftarrow UpdateBarycenter(\{\Phi_i | C_i = j\})$
end for$(B_j)_{j \leq k}$, $(P_i)_{i \leq n}$
return

Progressive Partial Bidding algorithm, we can enrich the input Persistence Diagrams with bidders of decreasing persistence in order to converge more rapidly and to stop the algorithm at a given time if needed.
The addition of new points in the Diagrams however creates a conflict with the use of lower and upper bounds in the Accelerated K-Means setup. When new points are added, upper and lower bounds on distances between input Persistence Diagrams and centroids need to be modified. However, thanks, one more time, to the triangle inequality, and to the fact that it is trivial to compute the distance between a diagram and a subdiagram of itself [7], these bounds can be updated without any difficulty.

Figures 4.11 and 4.12 in Section 4.2 show that the Progressive Accelerated K-Means tends to be faster than its classical accelerated counterpart, however, the progressivity tends to direct the clustering towards poor local minima, which is reflected in a loss of accuracy in the final clustering obtained.

[7] The optimal matching links each point of the first diagram that is absent in the second to the diagonal, the cost of the matching is therefore proportional to the sum of the squared persistences of the points that are in the first diagram but not in the second.

Experimental Results

4

Contents

This chapter describes the experiments made to validate the algorithms previously described, shows their results and tries to give some insights as to how the different variants of the algorithms differ in the way they work.

With the exception of results presented in Section 4.1.4, all computations were performed without any parallelism.

4.1 Fast Implementation of Barycenters

4.1.1 Auction and Use of KDTree

Our first experimental setup aims at validating the choice of the Auction over the Munkres algorithm. During the computation of barycenters with the Partial Bidding algorithm, we leverage the Auction in order to profit from its iterative nature. During the computation of distances, such a leverage is not possible. Nonetheless, the Auction coupled with the use of a KD-Tree can prove to be a way to reduce drastically the complexity of the matching algorithm.

In order to illustrate this fact, we created Persistence Diagrams using the following protocol :

1. Create two bidimensional sine waves on a square grid of size n^2.
2. Add white Gaussian noise to both signals.
3. Smooth the noisy signals using a flat kernel.
4. Compute the persistence diagrams of both scalar fields.

Different values of n yield Persistence Diagrams of different sizes since larger grids allow for more local extrema due to the presence of noise. For increasing values of n, we computed 100 pairs of Persistence Diagrams and we computed the 2-Wasserstein distance between them using the Auction algorithm and the Munkres algorithm.

Note that due to the random nature of the scalar fields used to compute them, the number of points in Persistence Diagrams obtained with a fixed value of n can vary. For each value of n tested, we chose to represent a single point on the graph, corresponding to the average number of points in the Persistence Diagrams obtained with this value of n and the average time needed to compute the distance.

Figure 4.1 shows the performance of the Munkres and Auction algorithms for the computation of the matching between two diagrams. It appears that the Munkres algorithm is faster for diagrams with fewer than 100 points each. However, for larger diagrams, its cubic complexity is beaten by the Auction. For large diagrams, one can easily see that the use of the K-D Tree counter-balances the cost of its creation and the Auction is faster with than without the K-D Tree, especially for large diagrams.

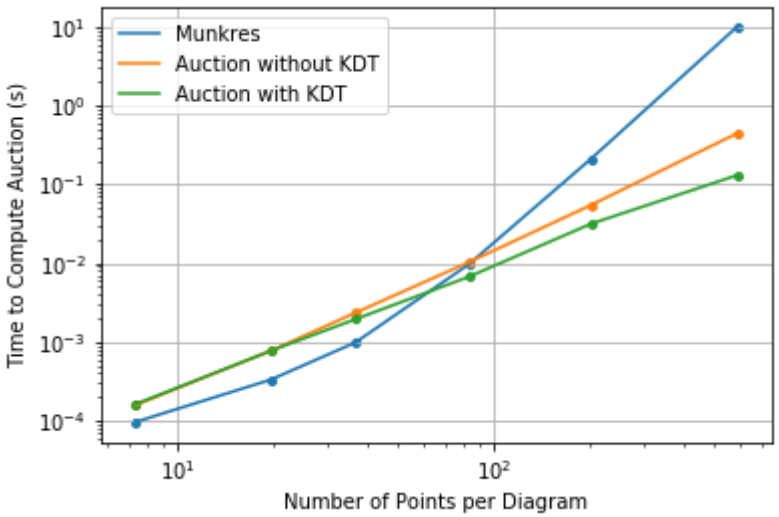

Figure 4.1 – *Time Performance of the Auction and Munkres algorithms for the Matching Computation (Auction precision :* 1%*)*

For diagrams of 600 points, the Auction with K-D Tree is more than 100 times faster than the Munkres algorithm and around 4 times faster than the Auction without KD-Tree.

4.1.2 Partial Bidding, price updates and ϵ-decrease

In this Section, we want to experimentally motivate the choice of the Partial Bidding strategy. To do so, we will compare the performances of the Partial Bidding with the performances obtained without ϵ-decrease, clever price initialization or with a full Auction between each barycenter update. We also aim at giving some insights

We generated sets of 10 Persistence Diagrams with the protocol described in Section 4.1.1. Each set of diagrams was obtained using 2 distinct frequencies for the Sine scalar field. 5 out of the 10 diagrams of each set had therefore more points than the 5 other.

The diagrams were generated for different values of n and we computed their barycenter using different sets of optimizations discussed in Section 3.1. In the sequel, we denote these optimizations as follows:

- (1) The ϵ-decrease optimization (ϵ decreases throughout the iterations of the algorithm, as opposed to ϵ being fixed at a small value),
- (2) The clever price initialization optimization,
- (3) The use of a single Auction round between each barycenter update, as opposed to the full computation of the matching with Auction Bidding between each barycenter update.

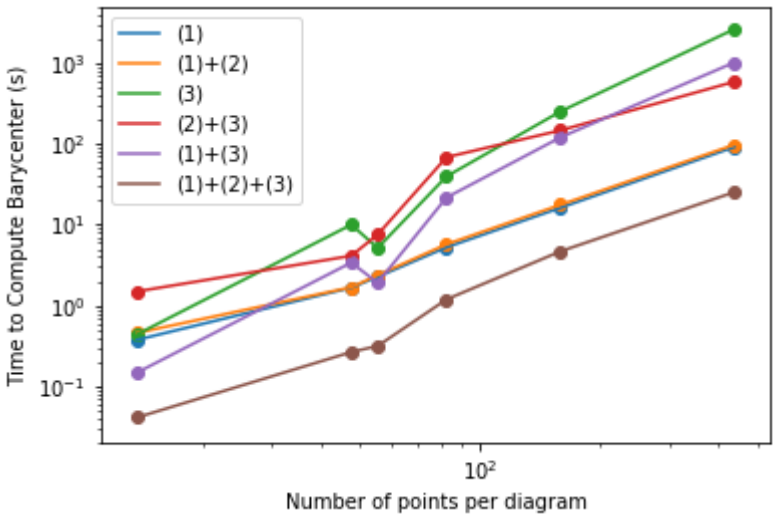

Figure 4.2 – *Time Performance of the Barycenter Computation for Dfferent Combinations of Optimizations*

Note that not using ϵ-decrease optimization has different meanings depending on whether or not optimization (3) is used.
With optimization (3), ϵ decrease means that ϵ changes between successive iterations of barycenter updates. Not using ϵ decrease means that ϵ is fixed to a small value and does not vary.
Without optimization (3), ϵ decrease simply means that ϵ decreases during the Auction Bidding algorithm. Not using decreasing ϵ in this situation has no meaning. Therefore, not using (1) only makes sense in the presence of a single Auction round (3).

We will denote each strategy used as a combination of these 3 optimizations: for instance $(1) + (2) + (3)$ is the Partial Bidding algorithm whereas (1) is the basic barycenter algorithm. Since either (1) or (3) need to be activated according to the remark above, there are six possible combinations of optimizations.

Figure 4.2 shows the comparative performances of the different combinations of optimizations on a log-log scale. One can observe that most combinations of optimizations perform worse than the basic algorithm.

Let's study the optimization (3) alone. With that strategy, between each barycenter updates, prices are forgotten and ϵ is set to a low value (in this case 2.10^{-5}). Low values of ϵ are efficient when the algorithm has already found prices close to their optimal values and that we are looking

for a precise matching. Of course, since prices are all set to 0 before the matching, this first assumption is erroneous. Moreover, as we stated in Section 3.1, first iterations of the barycenter algorithm do not need an overly precise matching since they will output approximations of the final barycenter. These remarks explain the poor performance of strategy (3).
Strategies (1) and $(1)+(2)$ have very similar time performances. This remark can be explained by the fact that the price information carried from one iteration to the following in strategy $(1)+(2)$ is not really taken advantage of. Indeed, the fact that ϵ_0, the value of ϵ at the first Auction Round between each barycenter update, is not decreasing discards most of the previous computations by allowing poorer matchings after the first Auction Round, leading to a total computational need equivalent to the one needed without price initialization.

The Partial Bidding strategy (or $(1)+(2)+(3)$) is the only one more performant than (1). This is particularly true for barycenter of diagrams with less than 100 points. For larger diagrams, the Partial Bidding approach seems to perform about 5 times faster than the approach (1).

The gain of time coming with the strategy $(1)+(2)+(3)$ is, in practice, *not* sacrificing the quality of the barycenter ouptutted. In our experiments, all 6 strategies led to similar costs or the final barycenter outputted. At most, there were differences of about 2% on the final barycenter's cost for different methods, but none of these diferences were statistically significant (p-value>0.05).

4.1.3 Progressive Barycenter

Section 3.2 explains how the Partial Bidding makes it possible to progressively add points to the diagrams with decreasing persistence. The progressivity allows not to take into account points with low persistence in early iterations of the Partial Bidding algorithm and add them later if necessary without restarting the computations from scratch.
This can be particularly useful if the user has a limited time bank to compute the barycenter. The computations can focus on persistent points in order to output a barycenter with precise position of its most persistent points but without low-persistence points which tend to carry less information about the objects from which the persistence diagrams were

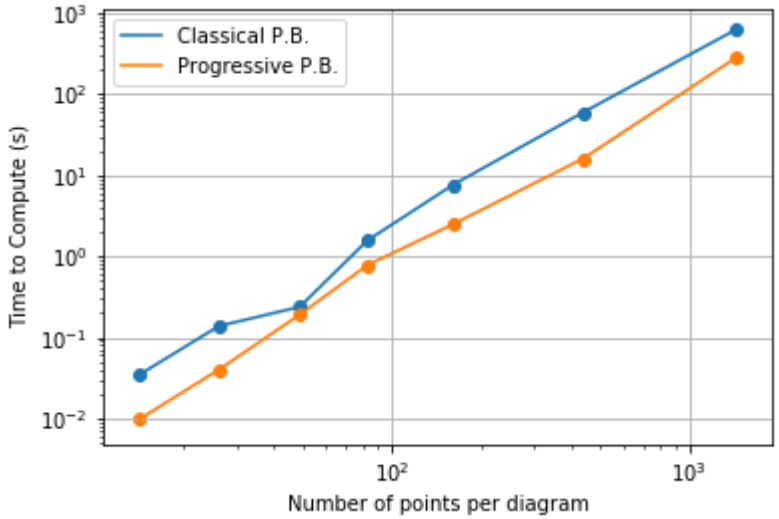

Figure 4.3 – *Time to Compute the Barycenter with or without the Progressivity of Partial Bidding*

computed.

We studied two quantities linked to the computation of the progressive barycenter:

- We compared the raw results of the progressive and non-progessive barycenter. We considered the time of computation as well as the final cost of the barycenter outputted.
- We studied the evolution of the number of points taken into account in the computation of the final barycenter as a function of the time limit chosen by the user (see Section 3.2.3 for a description of this time limit).

We show on Figures 4.3 and 4.4 the time performance of the progressive algorithm as well as the cost of the barycenter outputted.
It clearly appears that the progressive computation of the barycenter takes less time than the classical Partial Bidding method. We observed up to a factor 3 of difference in times of computation for diagrams of approximately 400 points. This difference may be explained by the early iterations of the non-progressive algorithm during which a lot of low-persistence points need to be matched whereas the progressive algorithm skips those points during the early iterations.

If the time performance of the Partial Bidding algorithm is improved when using the Progressive setup, the solutions it finds are more costly. Figure 4.4 shows that given sets of 10 diagrams, the Progressive Partial

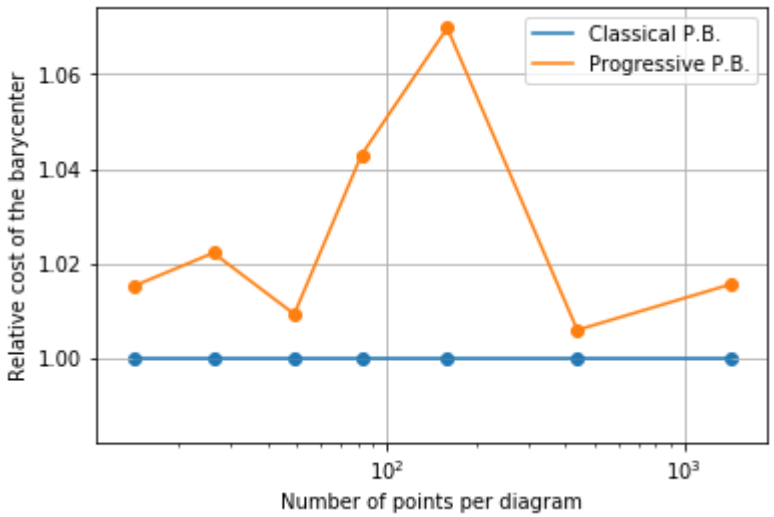

Figure 4.4 – *Final Cost of the Barycenter with or without the Progressivity of Partial Bidding*

Bidding finds barycenters with a higher cost than the classical Partial Bidding. The graph represents the relative cost of the Progressive Partial Bidding algorithm compared to the non-progressive: a relative cost of 1.02 means that on average, on identical sets of 10 diagrams, the progressive algorithm outputted barycenters with costs 2% higher.

We explained in Section 3.2.2 that using the Progressive Partial Bidding algorithm can easily allow the user to fix a time limit on the computation of the barycenter. One tenth of this time limit is used to progressively enrich the diagrams, the rest of it is used to let the algorithm converge (in a vast majority of the cases, the convergence is reach before the time limit). We show on Figure 4.5 the evolution of the final number of points used to compute the barycenter as a function of the user-defined time limit (*log-log* scale).
The y-axis shows the average number of points present in the 10 diagrams during the last iteration of the algorithm. The diagrams are computed using the protocol descibed in Section 4.1.1 and are initially composed of approximately 2500 points.
Obviously, when the time limit increases, the number of points taken in consideration also increases. Approximating the evolution to be polynomial, we found that the number of points used was roughly proportional to the square root of the time limit : $n_{points\ used} \approx A.\sqrt{t_{lim}}$.

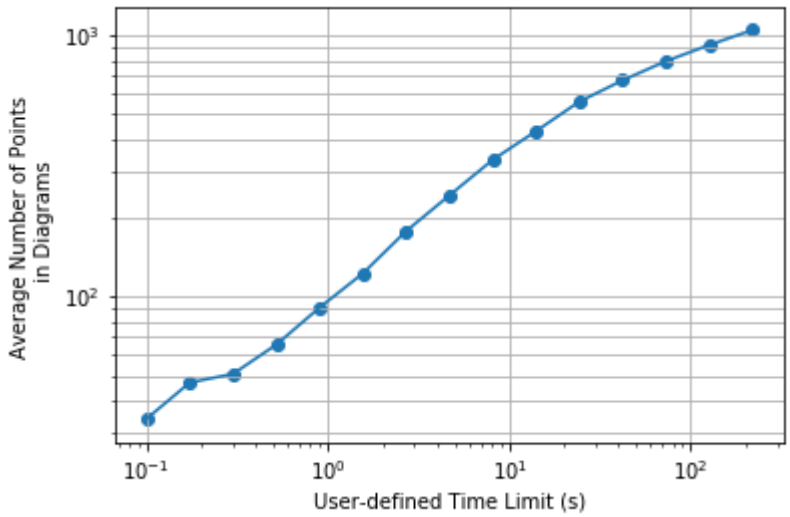

Figure 4.5 – *Average Number of Points used per the Diagram by the Progressive Partial Bidding as a Function of the Time Limit imposed on the Algorithm*

4.1.4 Parallelism

In this section, we study the effect of parallelism on the performance of the barycenter algorithm. To do so, we created sets of ten diagrams following the same protocol as in Section 4.1.2. We studied, for different number of threads used the time necessary to compute the barycenter.

With diagrams of similar size, we expect the speed-up to be approximately the number of threads as long as it does not get higher than the number of diagrams we want to compute the barycenter of. If the number of threads is higher than the number of diagrams, then, the expected speed-up is approximately the number of diagrams.

In reality, in the case of less diagrams than threads, the time of computation is determined by the following formula :

$$T_{threads>n_{diagrams}} = \sum_{Auction\ Round\ \alpha} \max_{i<n_{diagrams}} (T_\alpha(D_i))$$

Where $T_\alpha(D_i)$ is the time necessary to compute the matching between D_i and the barycenter at Auction Round α.

In our experiments, we computed the barycenters of sets of 10 diagrams. Each set of 10 diagrams can be divided in two halves, one of them contains diagrams with more points than the other half(cf. Section 4.1.2). In the limit case in which this half contains infinitely as many points as the other half, the computation of the matching to the barycenter for all

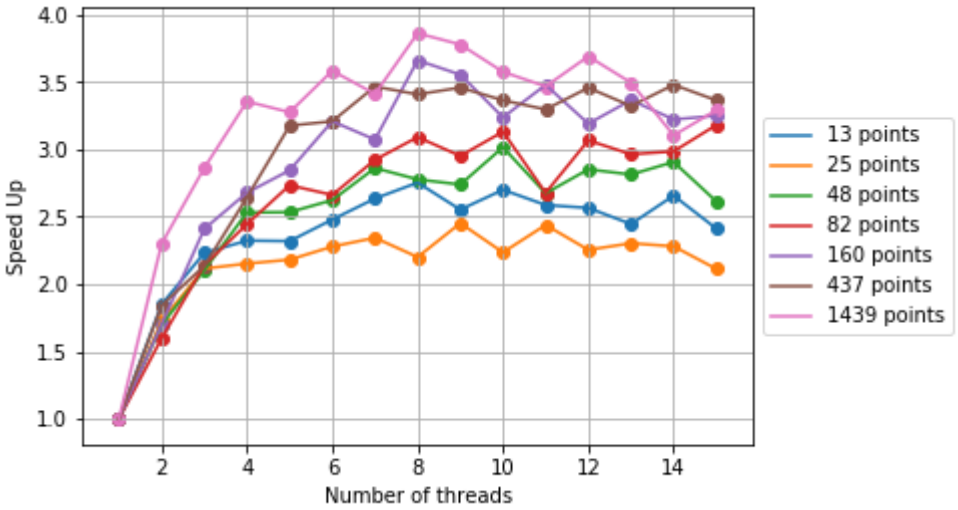

Figure 4.6 – *Speed-Up factor as a function of the average number of points in the diagramsand of the number of threads* (10 *diagrams of different sizes)*

diagrams would be, in terms of complexity, equivalent to the computation of the 5 matchings of the richest diagrams. We would therefore obtain a maximum speed-up of approximately 5.
In practice, the richest diagrams contain between 1.5 and and 2 times as many points as the poorest ones.

Some parts of the algorithm are still not parallelized (for instance, the update of the barycenter is one of them). Therefore, the effective speed-up of the algorithm is expected strictly to be below the factor 10 even in the perfect case in which all matchings take the same amount of time.

Figure 4.6 shows the evolution of the speed-up factor as the number of threads used increases.
It clearly appears that parallelism's efficiency increases as the number of points in each diagram increases. this behaviour can be explained by the fact that the parallelized part of the algorithm has a higher complexity than the sequential part. Therefore, the later becomes insignificant when the diagrams get more points, and as a consequence, the speed-up increaes.
The speed-up seems to be capped under 4 for large diagrams, and one can state that the marginal gain of a adding a new thread is more important when the number of thread is lower than or equal to 4 (which is logical since 5 out of the 10 matchings to be performed will out-cost the other 5 matchings), and that having more than 10 threads is useless.

4.1.5 Comparison with previous work

In this Section, we compare the performances of our algorithm with the algorithm of Lacombe et al. (2018). We used the same datata as in Section 4.1.4, namely 10 Persistence Diagrams obtained from bidimensional sine waves with two different frequencies.

For different sizes of diagrams we computed their barycenter using the Partial Bidding algorithm (without Progressivity) and with the Sinkhorn based algorithm introduced in Lacombe et al. (2018)[1]. We had to tune several parameters in the Sinkhorn algorithm, the principal ones being the value of γ, the coefficient for the entropic penalty and the stopping criterion. We chose to follow the suggestion of Lacombe for the choice of the entropice penalization and fixed it to $\gamma = \frac{10^{-1}}{p}$ where p is the average number of points in the diagrams, and the stopping criterion is met whenever the cost in two successive iterations does not decrease by more than 1%.
The choice of the size of the grid used for discretization is another problem. Due to the protocol used to compute the Persistence Diagrams, the diagrams with numerous points have the majority of their points are due to noise and are therefore of low persistence. If the discretization step is not small enough, most of these points are considered to be equal and are positionned on a square intersecting the diagonal. This means that points with low persistence will not be treated by Lacombe's algorithm if the discretization is not precise enough. In order to overcome this problem, we decided to run the experiment with the size of the grid on which all points with a persistence higher than 1% of the maximal persistence are not considered diagonal. This is achieved with a grid of size $d^2 = 100^2$. In order for the comparison to be fair, we therefore used diagrams with points of persistence higher than 1% of the highest persistence.

All experiments were performed on a CPU without any parallelization. We are fully aware that Lacombe's algorithm was designed to be GPU friendly and therefore, that the comparison with our performances is not entirely fair. Moreover, Lacombe's algorithm, due to its ease to be parallelized on GPU tends to be more efficient when dealing with a high number of Persistence Diagrams (the GPU can parallelize up to 1000

[1]We used the code that the authors kindly acceted to send us.

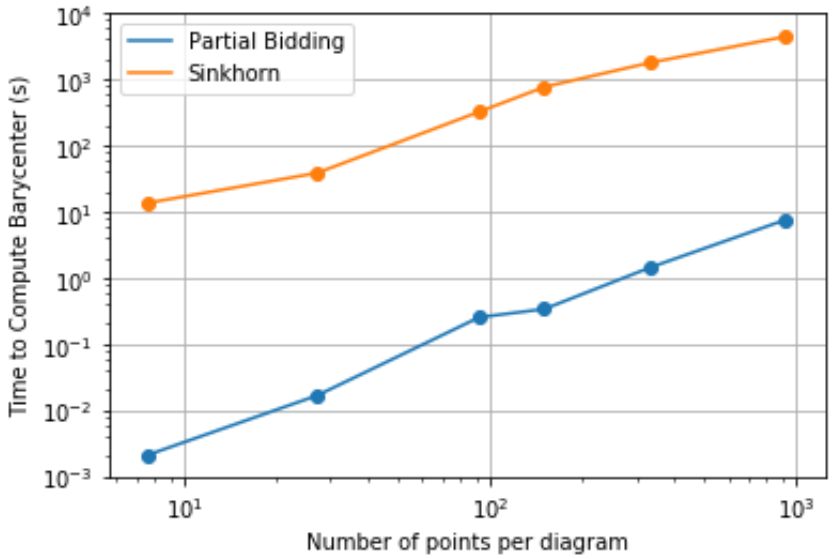

Figure 4.7 – *Time performances of the Partial Bidding algorithm and Lacombe's algorithm as a function of the size of the input Persistence Diagrams (log-log scale)*

diagrams at a time) that contain few points.

We show on Figure 4.7 the comparison of the performances of the Lacombe's algorithm and ours.

The graphs shows that, without parallelization or use of a GPU, with input 10 diagrams and with the choice of parameters described previously, the Partial Bidding algorithm outperforms the Sinkhorn algorithm by several orders of magnitude.

We think that this experiment along with our understanding of both algorithm show that the Partial Bidding algorithm and the Sinkhorn-based algorithm of Lacombe are complementary:

- The Partial Biddding algorithm has the advantage to be parameter-free and is faster when it is not parallelized or when it works on low number of diagrams.
  Moreover, the Partial Bidding algorithm outputs a Persistence Diagram whereas Lacombe's algorithm outputs a heatmap that is not trivial to analyze. The fact that the heatmap indicates approximate localizations for the points of the barycenter is not always enough: when dealing with Persistence Diagrams, each point corresponds to one precise topological feature in the data domain. Therefore, the cardinality of the diagram is important. Heatmaps do not allow the

user to know in a glimpse how many persistence pairs are represented in a given area of the diagram.

- On the other hand, Lacombe's algorithm can handle large scale computations on a GPU whereas the Partial Bidding algorithm has a linear complexity in the number of diagrams and cannot profit from the use of a GPU. Furthermore, the fact that the output of Lacombe's algorithm is a heatmap is not a problem in classification contexts since distances between Persistence Diagrams and the centroid heatmaps can be computed and therefore the K-Means algorithm can be applied.

As a conclusion, we would recommand to use Lacombe's algorithm for contexts, like the K-Means algorithm, where the barycenter does not need to be a Persistence Diagram, if a GPU is available, and when the number of Persistence Diagrams to consider is significant.
If the number of Persistence Diagrams is low, if a GPU is not available or if the context dictates that the output of the barycenter algorithm must be a Persistence Diagram (for visualizations purposes for instance), we believe that the Partial Bidding algorithm is more appropriate.

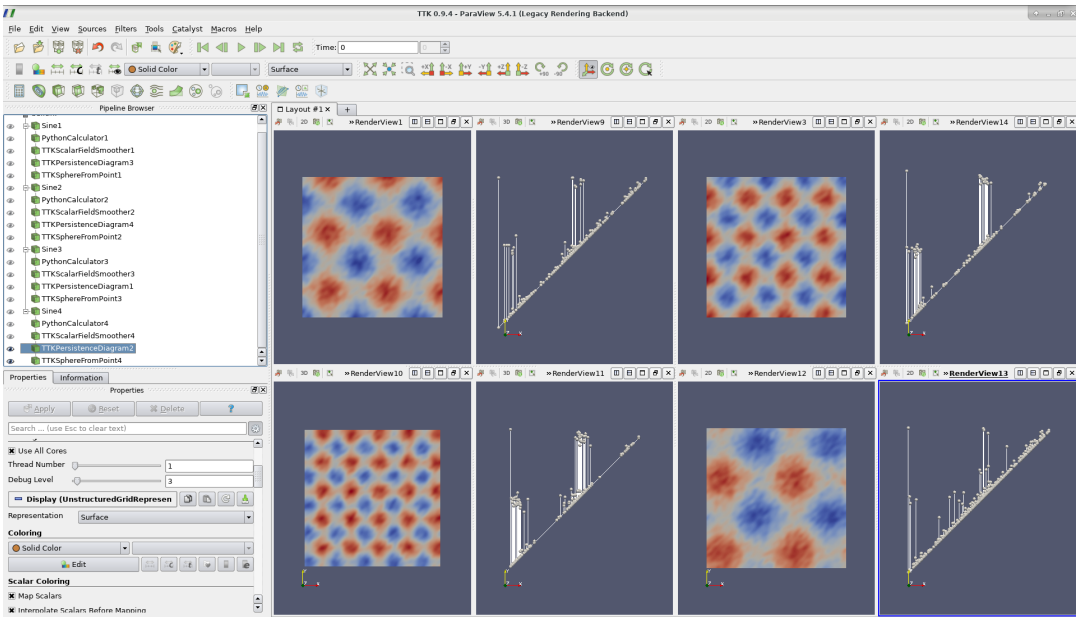

Figure 4.8 – *Scalar Fields and their Corresponding Persistence Diagrams*

4.2 KMeans Clustering of Persistence Diagrams

4.2.1 Computation on a simple example

In order to prove that our Partial Bidding K-Means algorithm is able to obtain a correct clustering, we created 4 sets of 5 diagrams each from 2-dimensional Sines of different frequencies to which white Gaussian noise and smoothing were added the same way as described in 4.1.1.

We show a screenshot from the TTK/Paraview software on Figure 4.8 displaying an example of scalar field and of the corresponding Persistence Diagram from each cluster.
Each diagram can be considered as a superposition of two different sub-diagrams. The first one contains the persistence pairs corresponding to the pairs of minimal and saddle points in the scalar field domain. The second one contains the persistence pairs corresponding to the pairs of saddle and maximum points in the scalar field domain. The matchings and the barycenter update are therefore computed independently (the centroids for each cluster are actually composed of two sub-diagrams, on for the minima-saddle pairs and one for the saddle-maxima points), however, the belonging of each diagram to a cluster is computed using the sum of distances between sub-diagrams.

On the example studied in this section, the diagrams contain between

68 and 119 points. There are 4 clear clusters that we aim at finding using the K-Means algorithm.

Using the K-Means++ initialization and the triangle inequality-based accelerated K-Means, and without parallelization, we found the correct clustering after 21 K-Means iterations. However the clustering was already correct after the first K-Means iteration. The following iterations were only used to refine the different centroids.

4.2.2 Use of Accelerated KMeans

In Section 3.4.1, we explained how the triangle inequality can help drastically reduce the number of distance computations when running the K-Means algorithm.

We used this Accelerated K-Means technique in our experiments. The Partial Bidding techniques proved useful in this optimization too since when updating the distance from a diagram D to its centroid C in order to find the closest centroid to D, one can use the previous computations -and in particular the prices obtained during the previous matching between D and C- in order not to restart the distance computation from scratch.
For the sake of simplicity in the code, we used this trick only when C was the centroid corresponding to the cluster of D.
Using previous prices to compute distances to other centroids would require to save in memory a lot of prices, and if C is not the centroid of D's cluster, it is not completely clear how to use the prices found during the latest matching of D and C, especially if C has changed a lot since that matching.

In this section, we quantitatively show the usefulness of using the Accelerated K-Means algorithm in our approach. To do so, we generated the same datasets as in Section 4.2.1 (20 diagrams from 2-dimensional sines with 4 different frequencies), and computed the clustering using both the classical K-Means alogrithm and its accelerated counterpart. The number of cluster was manually set to 4 in order to match the data. In order to simplify time comparisons, we decided not to use parallelism.

We show on Figures 4.9 and 4.10 the time performance and the accuracy of the clustering using both the accelerated and the basic algorithms.

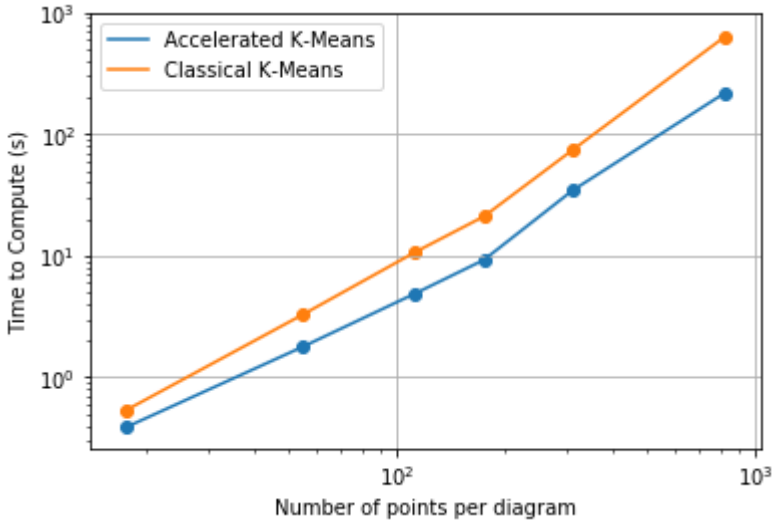

Figure 4.9 – *Time Performance of Clustering Algorithm with and without Triangle Inequality-based Acceleration (no parallelism)*

The Accelerated K-Means proved to be up to three times as fast as the non-Accelerated algorithm. This difference being higher with diagrams with high number of points.

Regarding the difference of accuracy, we stated in Section 3.4.1 that both algorithms are equivalent if distances are computed exactly. Since our matching algorithm is not exact, this assertion is theoretically false. However, in practice, both algorithm output similar accuracy as shown on Figure 4.10.

Note that the accuracy of the clustering is low for low number of points. This is not due to the algorithm itself, but to the input data, indeed, in order to obtain diagrams with few points, we generated scalar field on small grids. In particular, the smallest diagrams were obtained using a grid of 16×16 vertices. The lack of precision leads to seemingly very noisy Persistence Diagrams that are naturally difficult to tell apart.

This experiment justifies the use of the Accelerated K-Means algorithm since, for similar accuracies it saves more than half of the computation.

4.2.3 Progressive Clustering

In this section, we compare the performances of the Progressive clustering (using the Accelerated K-Means) described in Section 3.4.3 with the performances of the non-progressive Accelerated K-Means clustering. We

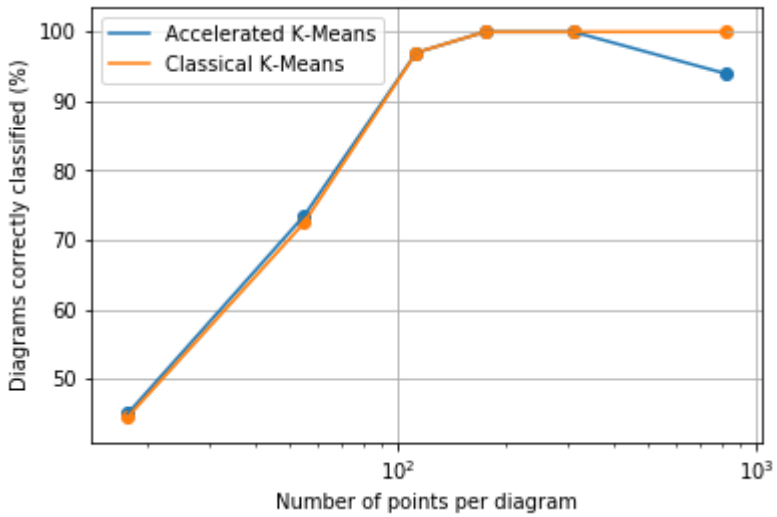

Figure 4.10 – *Accuracy of Clustering Algorithm with and without Triangle Inequality-based Acceleration*

used the data as in Section 4.2.2.

One of the greatest challenge encountered when performing the progressive clustering was the initialization. It is well known that a poor initialization of the K-Means algorithm often leads to local minima with high energy. In the previous sections, we intialized the K-Means process using a variant of the K-Means++ technique. However, doing so requires computation of distances between diagrams. In the progressive setup these distances are computed on partial diagrams and are therefore aproximations of the distances between the full diagrams. The approximation is better as the number of points in the diagrams increases. As a consequence; in order to use the *K-Means++* initialization, the choice of the initial number of points in each diagram (or equivalently the first persistence threshold) is crucial. We realized that fixing it to at most 10 points was largely insufficient as the final accuracy obtained was about 70% for large diagrams. We manually fixed it to 50 points for the purposes of this section.

The results of the experiments are shown on Figure 4.11 and 4.12. Figure 4.11 shows the time needed to converge for both Progressive and non-Progressivee K-Means algorithms. As the number of points increases, the progressive K-Means gets faster than its non-Progressive counterpart, finally getting up to 3 times as fast. Note that with diagrams of less than 50 points the progressive algorithm performs the same steps as the

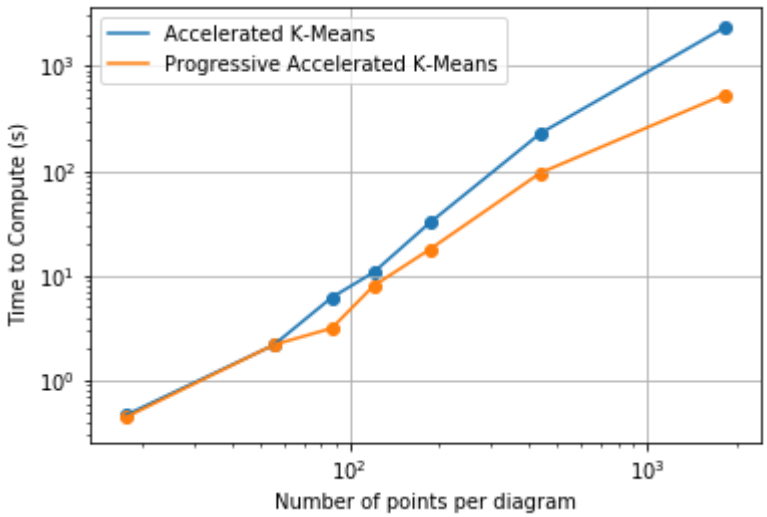

Figure 4.11 – *Time Performance of Clustering Algorithm with and without using Progressivity*

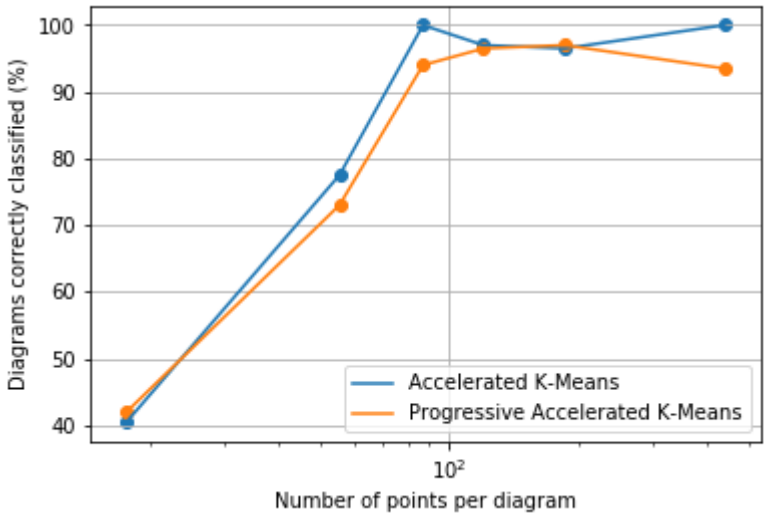

Figure 4.12 – *Accuracy of Clustering Algorithm with and without using Progressivity*

non-progressive one (except for the non-deterministic K-Means++ initialization), therefore, their time needed to converge are similar.

Figure 4.12 shows the accuracy of the algorithms. The progressive algorithm tends to be less precise, the cause was explained earlier: even though using initial diagrams with 50 points is better than with 10 points, the distances computed at the initialization step are still approximations. Sometimes, these approximations lead to a poor initial choice of centroids and to a poor clustering.

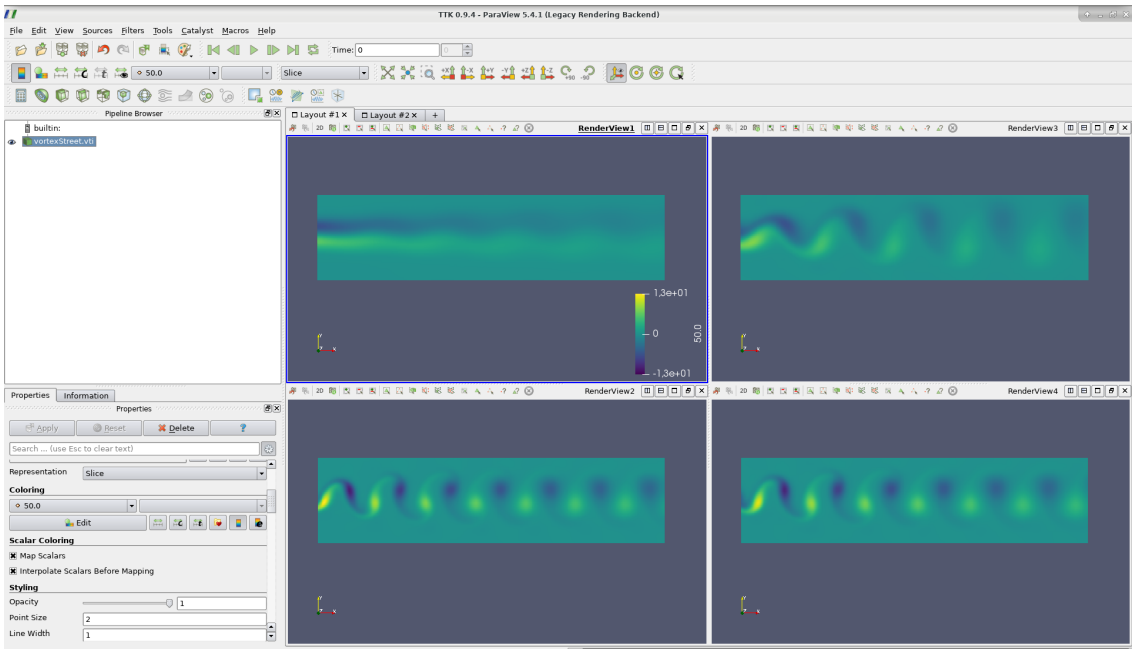

Figure 4.13 – *Vorticity Data : TTK/Paraview Screenshot of an Some of the Data*

4.3 Performance on Real Data

4.3.1 Presentation of the Data

In this section, we aim at proving that our approach can handle real datasets.

We used a dataset obtained from the simulation of a fluid flowing past a cylindric obstacle. This setup is known to create vortices that arrange themselves regularly in space in a phenomenon called *Karman vortex street*. Such vortices are an object of study in the engineering domain since they can induce unwanted vibrations of the obstacle. This causes problems of stability for towers under windy conditions or for planes during landing. The dataset consists of 45 scalar fields obtained from fluid dynamics simulations. A fluid is injected (from left to right) and encounters an obstacle. It creates a Karman vortex street. The dataset is split into 5 balanced classes corresponding to different viscosities of the fluid studied (and therefore different Reynolds numbers). Each class contains 9 scalar fields obtained for different time steps during the simulation.

We represented on Figure 4.13, one instance of 4 of the 5 classes of the dataset.

Our goal is to be able to regroup, the scalar fields corresponding to identical Reynolds numbers using the Persistence Diagrams obtained from them.

Persistence Diagrams are computed from the scalar fields, each diagram contains between 10 and 50 points. The clustering was performed on them using the Progressive Partial Bidding K-Means algorithm (with the acclerated strategy described in Section 3.4.1).
Using the K-Means++ initialization and using a time limit of 4 seconds, we were able to find the clusters with a 100% accuracy in 4.18s without any parallelization.
As a matter of comparison, with a random initialization, the accuracy drops to 86.7% in 5.74s of computation.

5 Future Work and Perspectives

Contents

This chapter describes some improvements that were thought of but not implemented and tested during the time of the internship. We believe that these approaches could be required in order to use our methods for larger datasets: Diagrams containing more points as well as a larger number of Persistence Diagrams.

5.1 Parallelism at Auction Level

In Section 3.3 we described how the Partial Bidding algorithm can be parallelized. We chose to compute the matching in each Auction Round in parallel Diagram wise. This choice was justified by the fact that the computation of the matching between each Persistence Diagram and the current barycenter are independent.
However, this approach has a major drawback: the number of threads used, and therefore, the maximal speed-up, can, at most, be equal to the number of Persistence Diagrams in the dataset. For datasets with few Persistence Diagrams but with many points in it, this is highly inefficient.

In order to overcome this difficulty, we suggest to add some parallelism at the level of the Auction. During an Auction Round bidders place bids on objects one after the other. If prices are set, these bids are indeppendant one from the other. However, placing a bid on an object changes the price of the object and affects the following bids.
Kerber et al. (2016) describes the Jacobi bidding procedure in which all unassigned bidders place bets simulatenously on their most valuable object. In case of conflict on an object, the bidder that proposed the highest price wins owns the object, the others stay unassigned. This strategy is less efficient than the iterative bidding since many conflicts can arise. Nevertheless, an intermediate strategy in which $k > 1$ bidders find their closest two neighbours in parallel and bid on them (after resolving conflicts the same way as in the Jacobi approach) could be time-saving, especially if bidders that bid simultaneously are geometrically far from each other since it would reduce the probability of a conflict.
We believe that performing a parallel Auction could enhance the speed-up of the barycenter computation if there are fewer Persistence Diagrams to consider than threads available. The problem of finding unassigned bidders far from each other does not seem to be easy to solve, but choosing them at random should be efficient enough.

5.2 Discretization and Multi-Bidder Auction for Large Diagrams

Our experiments showed that working with Persistence Diagrams with large number of points (3000 or more) led to prohibitive computation times. Kerber et al. (2016) suggested to discretize Persistence Diagrams

in order to compute an approximation of their distances. This approach means that bidders could have a multiplicity $p > 1$ with non-neglictable probability, even in real datasets, they would therefore share the same closest objects in the barycenter. This can be used to compute more efficiently the optimal matching through Auction Bidding. Bidders with multiplicity $p > 1$ could bid on p objects simultaneously, this could reduce the number of searches for closest neighbours and therefore the complexity of the algorithm.
We did not perform any test with this method but it nonetheless seems promising. In particular, refining the discretization as the Partial Bidding algorithm runs could make the first iterations faster and still output precise results at the end. Lacombe et al. (2018) used discretization on a regular grid of the Persistence Diagrams before computing their barycenter. They showded that the loss of precision induced by the discretization can easily be bounded.

5.3 Multiple Partial Bidding

The main idea of the Partial Bidding algorithm is to avoid the redundancy of computing all Auction Rounds from the scratch each time a matching must be found between two Persistence Diagrams. We chose to use the most extreme choice of performing a single Auction Round between two barycenter updates, but there is no reason for this choice to be optimal.
Performing $k > 2$ Auction Rounds between each barycenter update with a higher value of ϵ for the first round could prove to be more efficient in certain cases.

We leave the choice of k as an open problem that seems promising to us.

5.4 Online Barycenter and K-Means

We introduced in Sections 3.2 and 3.4.3 an algorithm that progressively adds points of decreasing persistence to the Persistence Diagrams in order to make the algorithm faster and to allow the user to choose a time limit to the computations. This is particularly adapted for contexts with high time constraints.
In online contexts, like when time-dependant phenomenons are simulated, Persistence Diagrams may be added to the dataset in real time.

Adapting the Partial Bidding methods to add new input diasgrams could vastly help.

For the computation of barycenters, the addition of a new Persistence Diagram D_{new} during the execution of the Partial Bidding algorithm could be managed similarly to changes of cluster in the Partial Bidding K-Means algorithm: D_{new} is matched to the current barycenter using Auction Bidding until a sufficient precision is achieved (when the ϵ parameter of the Auction Bidding get lower than the current value of ϵ in the barycenter algorithm). At this point, the prices for D_{new} are precise enough and the Partial Bidding algorithm can run its course.

Regarding the Partial Bidding K-Means algorithm the addition of new Persistence Diagram in the dataset would simply require to find its closest centroid before letting the algorithm continue.

A risk of these methods is that starting computations with a partial dataset may lead to local minima with higher cost than what would be obtained using the whole dataset from the start of the algorithm.

Conclusion

6

This manuscript describes my work on the topic of Wasserstien Barycenters of Persistence Diagrams. In particular, it introduces two major algorithms, the Partial Bidding algorithm and the Partial Bidding K-Means algorithm, that respectively compute the barycenter of a set of Persistence Diagrams and its clustering. It also introduces their progressive versions that allows the user to stop the computations at any time given in advance and still obtain meaningful results that have converged. The computational crux of computing the Wasserstein barycenter of Persistence Diagrams is the computation of optimal transportation plans. In that respect, Bertsekas Auction algorithm helped us a lot and is the basis of all the work presented in this document.

We compared the performances of our algorithm to the performances of previously implemented algorithms and showed that the Partial Bidding algorithm is faster than previous State of the Art methods. We believe that our algorithms can be complementary to methods based on entropic penalization like the ones introduced by Lacombe, since they are adapted to very different contexts.
Our results show that the progressive version of both algorithm reduces their execution time although for the K-Means algorithm it comes at the price of a lower accuracy.

We believe that some more research is still needed to improve the algorithm. Chapter 5 presented some ideas to enhance the Partial Bidding algorithm, but the list is not exhaustive.

Bibliography

Adams, H., Emerson, T., Kirby, M., Neville, R., Peterson, C., Shipman, P., Chepushtanova, S., Hanson, E., Motta, F., and Ziegelmeier, L. (2017). Persistence images: A stable vector representation of persistent homology. *The Journal of Machine Learning Research*, 18(1):218–252. (Cited page 14.)

Bauer, U., Ge, X., and Wang, Y. (2013). Measuring distance between reeb graphs. *CoRR*, abs/1307.2839. (Cited pages 12 and 13.)

Bertsekas, D. P. (1981). A new algorithm for the assignment problem. *Mathematical Programming*, 21(1):152–171. (Cited pages 17 and 20.)

Bubenik, P. (2015). Statistical topological data analysis using persistence landscapes. *The Journal of Machine Learning Research*, 16(1):77–102.

Celebi, M. E., Kingravi, H. A., and Vela, P. A. (2012). A comparative study of efficient initialization methods for the k-means clustering algorithm. *CoRR*, abs/1209.1960. (Cited page 48.)

Celebi, M. E., Kingravi, H. A., and Vela, P. A. (2013). A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert systems with applications*, 40(1):200–210.

Cole-McLaughlin, K., Edelsbrunner, H., Harer, J., and Natarajan, V. (2003). Loops in reeb graphs of 2-manifolds. 32. (Cited page 14.)

Cuturi, M. (2013). Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pages 2292–2300. (Cited page 15.)

Cuturi, M. and Doucet, A. (2014). Fast computation of wasserstein barycenters. In *International Conference on Machine Learning*, pages 685–693. (Cited page 16.)

Dey, T., Shi, D., and Wang, Y. (2015). Comparing graphs via persistence distortion. (Cited page 13.)

Edelsbrunner, H. and Harer, J. (2010). *Computational topology: an introduction*. American Mathematical Soc. (Cited page 26.)

Elkan, C. (2003). Using the triangle inequality to accelerate k-means. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pages 147–153. AAAI Press. (Cited pages 46 and 47.)

Favelier, G., Faraj, N., Summa, B., and Tierny, J. (2018). Persistence atlas for critical point variability in ensembles. *arXiv preprint arXiv:1807.11212*. (Cited page 27.)

Ferstl, F., Burger, K., and Westermann, R. (2015). Streamline variability plots for characterizing the uncertainty in vector field ensembles. 22. (Cited page 26.)

Ferstl, F., Kanzler, M., Rautenhaus, M., and Westermann, R. (2016). Visual analysis of spatial variability and global correlations in ensembles of iso-contours. *Computer Graphics Forum*, 35(3):221–230. (Cited page 26.)

Gunther, D., Salmon, J., and Tierny, J. (2014). Mandatory critical points of 2d uncertain scalar fields. 33:31–40. (Cited page 26.)

Hilaga, M., Shinagawa, Y., Komura, T., and Kunii, T. (2001). Topology matching for fully automatic similarity estimation of 3d shapes. pages 203–212. (Cited page 13.)

Kerber, M., Morozov, D., and Nigmetov, A. (2016). Geometry helps to compare persistence diagrams. *CoRR*, abs/1606.03357. (Cited pages 8, 17, 22, 23, and 75.)

Lacombe, T., Cuturi, M., and Oudot, S. (2018). Large Scale computation of Means and Clusters for Persistence Diagrams using Optimal Transport. *NIPS 2018*. (Cited pages 16, 26, 38, 63, and 76.)

Munch, E., Turner, K., Bendich, P., Mukherjee, S., Mattingly, J., Harer, J., et al. (2015). Probabilistic fréchet means for time varying persistence diagrams. *Electronic Journal of Statistics*, 9(1):1173–1204. (Cited page 26.)

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38. (Cited page 8.)

Parsa, S. (2013). A deterministic o(m log m) time algorithm for the reeb graph. 49:864–878. (Cited page 12.)

Reeb, G. (1946). Sur les points singuliers d'une forme de pfaff completement integrable ou d'une fonction numerique [on the singular points of a completely integrable pfaff form or of a numerical function]. *Comptes Rendus Acad. Sciences Paris*, 222:847–849. (Cited page 12.)

Saikia, H., Seidel, H., and Weinkauf, T. (2014). Extended branch decomposition graphs: Structural comparison of scalar data.

Soler, M., Plainchault, M., Conche, B., and Tierny, J. (2018a). Lifted wasserstein matcher for fast and robust topology tracking. *arXiv preprint arXiv:1808.05870*. (Cited page 37.)

Soler, M., Plainchault, M., Conche, B., and Tierny, J. (2018b). Topologically controlled lossy compression. *arXiv preprint arXiv:1802.02731*.

Turner, K., Mileyko, Y., Mukherjee, S., and Harer, J. (2012). Fréchet Means for Distributions of Persistence diagrams. *ArXiv e-prints*. (Cited pages 8 and 17.)