Topological Optimization with Big Steps

Arnur Nigmetov Dmitriy Morozov
Lawrence Berkeley National Laboratory

November 19, 2023

Abstract

Using persistent homology to guide optimization has emerged as a novel application of topological data analysis. Existing methods treat persistence calculation as a black box and backpropagate gradients only onto the simplices involved in particular pairs. We show how the cycles and chains used in the persistence calculation can be used to prescribe gradients to larger subsets of the domain. In particular, we show that in a special case, which serves as a building block for general losses, the problem can be solved exactly in linear time. This relies on another contribution of this paper, which eliminates the need to examine a factorial number of permutations of simplices with the same value. We present empirical experiments that show the practical benefits of our algorithm: the number of steps required for the optimization is reduced by an order of magnitude.

1 Introduction

Topological optimization [18, 26, 10] is a novel application of persistent homology [20]. The basic idea is to define a loss in terms of the points of a persistence diagram and minimize it using the modern optimization software that combines automatic differentiation with state-of-the-art optimization techniques. Depending on the application, we may want to reduce noisy features in the data by moving low-persistence points closer to the diagonal [26, 25] or outside of a particular quadrant [10], amplify signal by moving high-persistence points away from the diagonal [7], match a template signal by moving the current diagram towards a prescribed one [18, 22], among other applications. Most of the work so far has been motivated by problems in machine learning: a loss formulated via persistence can be used to regularize a decision boundary (following the philosophy that overfitting produces a topologically complex surface).

Optimization also offers a new approach to an old problem. Given a function $f : \mathbb{X} \to \mathbb{R}$ on some topological space, persistence-sensitive simplification [16] asks for a nearby function $g : \mathbb{X} \to \mathbb{R}$, with the same persistence diagram as f, but without the points closer than ε to the diagonal. The original paper [16] showed that one can solve the problem for extrema — and therefore, by duality, completely on 2-manifolds — but the suggested algorithm was ad hoc. The running time was later improved to linear [2, 5], see also [29]. Crucially, the problem has only been solved for extrema, with the difficulty of processing middle dimensions (e.g., simplifying 1-dimensional persistence for functions on 3-manifolds) highlighted by the connection to the Poincaré conjecture [24, Section 3.5]: because simplification is impossible for sufficiently large values of ε on homology spheres, any such scheme must take the topology of the domain into account.

But it is possible to take a "best effort" approach. Instead of solving the problem exactly via combinatorics, one can formulate a simplification loss that penalizes points closer to the diagonal than ε. Minimizing such a loss may not produce the perfect solution g, but it can get very close. Moreover, it offers the flexibility of articulating more sophisticated goals: for example, to not only simplify the function overall, but also to control the topology of its specific levelsets or sublevel sets.

Approach. We are interested in the general problem, where the loss is formulated as a partial matching. Some of the points p_i in the persistence diagram are prescribed targets q_i, and the loss aims to minimize

the distance between them, e.g., $\mathcal{L} = \sum_i (p_i - q_i)^2$. The existing approaches to this optimization are all based on the same idea. Each point in the diagram is defined by the values of a pair of simplices: $p_i = (b_i, d_i) = (f(\sigma_i), f(\tau_i))$, where $f : K \to \mathbb{R}$ is the input filtration. The gradient $\partial\mathcal{L}/\partial p_i$ defined by the loss immediately translates to the gradient on the simplex values, $\partial\mathcal{L}/\partial f(\sigma_i)$ and $\partial\mathcal{L}/f(\tau_i)$. These in turn can be backpropagated through the filtration to define the gradients on the input data.

This approach is general — it can handle arbitrary losses — and comes with theoretical guarantees of convergence [8]. But it is also slow. Subsampling [28] has been suggested as a way to speed it up. We instead improve performance by focusing on one of its shortcomings, namely that it treats persistence as a black box. The only information used comes from the pairing, which means that only critical values of the input get any gradient information: each point p_i gives gradients on only two simplex values. Moreover, if the optimization is done carefully and multiple simplices get the same value, the above gradient definition is not even correct: defining it exactly in general requires examining $k!$ different orders of the k simplices with the same value [22].

At the same time, persistent homology computes a lot more structure than just the pairing represented in the persistence diagram. The standard algorithms [20, 13] compute cycles and chains in the domain that certify the existence of a particular pair. We take advantage of this extra information to speed up optimization by suggesting a principled way for each point to define gradients for a large set of simplex values.

Our work has four main **contributions**:

1. We show that for a simple loss, called *singleton loss*, defined by matching a single point in the persistence diagram to a target, the gradient can be computed exactly, including when multiple simplices have the same value, by examining a single permutation, rather than $k!$ required in general. This structural realization leads to a cubic algorithm to optimize the singleton loss.
2. We show that this algorithm can be improved to linear time by examining matrices computed as a byproduct of finding the persistence pairing.
3. We introduce a set of heuristics for combining "big steps" prescribed by individual points into a gradient on both critical and regular simplices, which can be backpropagated and optimized using standard algorithms and software.
4. We show experimentally that our procedure requires an order of magnitude fewer steps to optimize a loss than the standard procedure that defines the gradient on only two simplices per persistence pair.

2 Background

We assume the reader's familiarity with algebraic topology and only briefly review the setting of persistent homology, to establish the notation. We refer the reader to the extensive resources [14, 15] for a thorough introduction.

Persistent homology. Given a simplicial complex K, with n simplices, and a function $f : K \to \mathbb{R}$ that respects the face relation — i.e., $f(\sigma) \leq f(\tau)$ if σ is a face of τ — we sort the simplices in K by function value, breaking ties if necessary so that faces come before their cofaces. We use $<$ to denote the resulting total order on the simplices. We denote the subcomplexes defined by the prefixes of this order with K_i. Their nested sequence is called a *filtration*:

$$K_1 \subseteq K_2 \subseteq \ldots \subseteq K_n = K.$$

Using coefficients in a field and passing to homology, we get a sequence of homology groups, connected by linear maps induced by the inclusions:

$$\mathsf{H}_*(K_1) \to \mathsf{H}_*(K_2) \to \ldots \to \mathsf{H}_*(K_n).$$

Persistent homology tracks how classes appear and disappear in this sequence, and produces a set of pairs (σ_i, σ_j) such that a homology class created by simplex σ_i dies when simplex σ_j enters the filtration, and a set of infinite pairs (σ_i, ∞), if a class created by simplex σ_i does not die.

To compute this pairing, we start with the boundary matrices, D_p, of the simplicial complex, whose columns and rows are ordered by the filtration. Each such matrix stores the boundaries of the p-simplices.[1] It will be convenient to use the simplices themselves to index the columns of various matrices, so for example $D_p[\cdot, \tau]$ refers to the column that stores the boundary of p-simplex τ; similarly, $D_p[\sigma, \cdot]$ refers to the row that stores the coboundary of $(p-1)$-simplex σ.

Persistence pairing is computed by reducing the boundary matrix, which can be interpreted [11] as finding decompositions $R_p = D_p V_p$, where matrices R_p are reduced, meaning the lowest non-zeros in their columns appear in unique rows, and matrices V_p are invertible upper-triangular. There are many such decompositions — Algorithm 1 is the original algorithm [20] that finds one of them — but the locations of the lowest non-zeros in matrices R_p are unique and give the persistence pairing. Denoting by $\operatorname{low} R_p[\cdot, \tau]$ the simplex that corresponds to the row of the lowest non-zero entry in the column, we have a pair (σ, τ) iff $\operatorname{low} R_p[\cdot, \tau] = \sigma$ and a pair (σ, ∞) iff $R_p[\cdot, \sigma] = 0$ and there is no column with $\operatorname{low} R_p[\cdot, \tau] = \sigma$. We call such σ *positive* or *birth* simplices, and such τ *negative* or *death* simplices.

As in [11], we denote by U_p the inverse of matrix V_p, so that $D_p = R_p U_p$. Algorithm 1 shows how to compute matrices R_p, V_p, U_p. The columns of R_p and V_p have a natural interpretation: matrix R_p stores the cycles that generate the homology classes in the respective subcomplexes. Matrix V_p stores the chains that turn those cycles into boundaries.

Remark. *Columns of matrices D_p and R_p are indexed by the p-simplices; their rows, by the $(p-1)$-simplices. Both rows and columns of matrices V_p and U_p are indexed by the p-simplices.*

Algorithm 1 Lazy reduction of the boundary matrix.

```
R_p = D_p, V_p = I, U_p = I for all p
for all τ_j ∈ K (in filtration order) do
    while R_p[·, τ_j] ≠ 0 and ∃ τ_i < τ_j, low R_p[·, τ_i] = low R_p[·, τ_j] do
        σ = low R_p[·, τ_j]
        α = R_p[σ, τ_j] / R_p[σ, τ_i]
        R_p[·, τ_j] = R_p[·, τ_j] − α · R_p[·, τ_i]
        V_p[·, τ_j] = V_p[·, τ_j] − α · V_p[·, τ_i]
        U_p[τ_i, ·] = U_p[τ_i, ·] + α · U_p[τ_j, ·]
              (equivalently, U_p[τ_i, τ_j] = α)
```

Matrices U_p and V_p obtained via the lazy reduction in Algorithm 1 have a special property that we rely on below. Throughout the paper — starting from the statement and proof of the following lemma — it is convenient to simplify the language by assuming that if $R_p[\cdot, \tau] = 0$, then $\operatorname{low} R_p[\cdot, \tau]$ is implicitly equal to a "dummy" simplex $\bar{\sigma}$ that precedes every other simplex in the filtration order.

Lemma 1 (Lazy reduction)**.** *If decompositions $R_p = D_p V_p$ and $D_p = R_p U_p$ are obtained via the lazy reduction in Algorithm 1, then if $\sigma_i = \operatorname{low} R_p[\cdot, \tau_i]$ and $\sigma_j = \operatorname{low} R_p[\cdot, \tau_j]$ are such that $\tau_i < \tau_j$ and $\sigma_i < \sigma_j$, then $U_p[\tau_i, \tau_j] = V_p[\tau_i, \tau_j] = 0$.*

Proof. The proof is by induction. The statement is trivially true initially, when $V_p = U_p = I$. Suppose the statement is true after $l-1$ steps of the reduction. Suppose in step l we are adding a multiple of column $R_p[\cdot, \tau_i]$ to $R_p[\cdot, \tau_j]$. Since the reduction is lazy, it means $\operatorname{low} R_p[\cdot, \tau_i] = \operatorname{low} R_p[\cdot, \tau_j]$ before the addition, and $\operatorname{low} R_p[\cdot, \tau_j] < \operatorname{low} R_p[\cdot, \tau_i]$ afterwards. The corresponding operation in matrix V_p adds a multiple of column $V_p[\cdot, \tau_i]$ to column $V_p[\cdot, \tau_j]$, so the only non-zero entries that may be introduced into the column $V_p[\cdot, \tau_j]$ are those in the column $V_p[\cdot, \tau_i]$. By induction all of them fall in rows τ_k with $\operatorname{low} R_p[\cdot, \tau_k] \geq \operatorname{low} R_p[\cdot, \tau_i] > \operatorname{low} R_p[\cdot, \tau_j]$. Since by the time we are reducing $R_p[\cdot, \tau_j]$, we have already reduced all the preceding columns — and therefore their pairs don't change — the claim follows for matrix V_p.

[1] Recall that a p-simplex has $(p+1)$ vertices.

In matrix U_p, the corresponding operation is adding a multiple of row $U_p[\tau_j, \cdot]$ to row $U_p[\tau_i, \cdot]$. By induction any non-zero in the former falls in the columns τ_k with $\operatorname{low} R_p[\cdot, \tau_k] \leq \operatorname{low} R_p[\cdot, \tau_j] < \operatorname{low} R_p[\cdot, \tau_i]$. Since the already reduced columns in R_p don't change, the claim follows for matrix U_p. □

The following two corollaries follow immediately as contrapositive statements of the lemma. In both, because matrix R_p is reduced, the equality among the lowest entries is achieved iff $\tau_i = \tau_j$.

Corollary 2. *If after a lazy reduction entry $U[\tau_i, \tau_j] \neq 0$, then $\operatorname{low} R_p[\cdot, \tau_j] \leq \operatorname{low} R_p[\cdot, \tau_i]$.*

Corollary 3. *If after a lazy reduction entry $V[\tau_i, \tau_j] \neq 0$, then $\operatorname{low} R_p[\cdot, \tau_j] \leq \operatorname{low} R_p[\cdot, \tau_i]$.*

Duality. Passing from the filtration to cohomology, a vector space dual of homology, we get a sequence of cohomology groups, connected by linear maps induced by restrictions:

$$\mathsf{H}^*(K_1) \leftarrow \mathsf{H}^*(K_2) \leftarrow \ldots \leftarrow \mathsf{H}^*(K_n).$$

By duality [13], the pairing in this sequence is the same as for homology, but with the role of birth and death reversed, a fact we exploit below.

Algorithmically, we replace the boundary matrix by its anti-transpose, $D_p^\perp$, i.e., a transpose of D_p with rows and columns ordered in reverse filtration order. Applying Algorithm 1, we get decompositions $R_p^\perp = D_p^\perp V_p^\perp$ and $D_p^\perp = R_p^\perp U_p^\perp$. Similar to homology, the matrices have immediate interpretation: $R_p^\perp$ stores the cocycles and $V_p^\perp$ the cochains that turn them into coboundaries.

Remark. *Matrices $R_p^\perp, V_p^\perp, U_p^\perp$ are not anti-transposes of matrices R_p, V_p, U_p. In $D_p^\perp$ and $R_p^\perp$, rows are indexed by p-simplices; columns, by $(p-1)$-simplices. In $V_p^\perp$ and $U_p^\perp$, both rows and columns are indexed by $(p-1)$-simplices.*

Persistence pairing is the same for homology and cohomology [13]. $\operatorname{low} R_p[\cdot, \tau] = \sigma$ iff $\operatorname{low} R_p^\perp[\cdot, \sigma] = \tau$ (simplices σ and τ are paired). $R_{p-1}[\cdot, \sigma] = 0$ and $\nexists \tau$ with $\operatorname{low} R_p[\cdot, \tau] = \sigma$ iff $R_p^\perp[\cdot, \sigma] = 0$ and $\nexists \rho$ with $\operatorname{low} R_{p-1}^\perp[\cdot, \rho] = \sigma$ (simplex σ is unpaired).

Stability. In the combinatorial setting, the following statement is equivalent [11] to the stability of persistent homology.

Lemma 4. *Suppose two simplices σ_1 and σ_2 that appear consecutively in the filtration transpose. The only persistence pairs that can change are the two pairs that involve simplices σ_1 and σ_2.*

It follows immediately that re-ordering more than two simplices only affects their respective pairs.

Corollary 5. *Given a contiguous set of simplices X in the filtration, changing the order of simplices in X can change only persistence pairs with one of the endpoints in set X.*

3 Singleton Loss

Virtually every topological loss proposed in the literature can be rephrased as a partial matching: some points in the diagram are prescribed targets, where they need to move. For example, the simplification loss,

$$\mathcal{L}_\varepsilon(f) = \sum_{\substack{(b,d) \in \operatorname{Dgm}(f) \\ (d-b) \leq \varepsilon}} (d-b)^2 \tag{1}$$

can be formulated as a partial matching M, where every point $(b, d) \in \operatorname{Dgm}(f)$ with $(d-b) \leq \varepsilon$ is matched to the point $((b+d)/2, (b+d)/2)$. Then the loss can be re-written as

$$\mathcal{L}_\varepsilon(f) = \sum_{(p,q) \in M} (p-q)^2.$$

We consider the simplest such setting, where the partial matching consists of a single pair, (p, q). We call this *singleton loss*. We assume that $p = (b, d) = (f(\sigma), f(\tau))$ and $q = (b', d')$. The loss itself is $\mathcal{L} = (p - q)^2$. We will follow the gradient flow of this loss and keep track of point p. Specifically, we denote by p_t the image of p under gradient flow after time t, and write the loss $\mathcal{L}_t = (p_t - q)^2$, which defines the gradient at every point in time. We note that the loss is oblivious to what happens to the other points in the persistence diagram.

Following the gradient to minimize this loss translates into moving simplices σ and τ in the filtration to their target values b' and d'. We first focus on the negative p-simplex τ. Suppose there are m simplices with values between d and d', and m_p of them are p-simplices. As we increase or decrease the value of τ (depending on whether $d' > d$ or $d' < d$), it is going to reach the value of each one of the m_p p-simplices. For each such simplex τ_k, we must determine what happens if we place it before τ, when increasing the value, or after τ, when decreasing. If doing so changes the pairing of σ to τ_k, then τ_k needs to move together with τ (and enter the *critical set* X_σ, defined below). If not, we can safely skip over τ_k (a fact that itself requires a proof).

It is not immediately obvious, but we prove in the next subsection that when determining the fate of τ_k, it is not necessary to consider all $k!$ possible orders of the simplices that are moving together with τ (as one might reasonably expect in general [22]); determining the pairing for a single order suffices.

Besides moving the simplices of the same dimension as τ, which are the only simplices that may take over the pairing with σ, we must move all of their cofaces, when increasing the value, or their faces, when decreasing the value. This is required simply to ensure that our simplex order defines a filtration. We revisit this topic in Section 3.6.

3.1 Critical Set

As we move a p-simplex τ, paired with a q-simplex σ,[2] in the filtration, we maintain a critical set of p-simplices that move together with τ under the gradient flow of the singleton loss. We say that a set of p-simplices is *contiguous*, if their columns are contiguous in matrix D_p.

Definition 6. *Given a q-simplex σ, a set of contiguous p-simplices X_σ is* critical, *if placing any $\tau' \in X_\sigma$ as the first simplex (when increasing the value of τ) or as the last simplex (when decreasing the value of τ) in the set makes it paired with σ.*

The critical set is well-defined because whether simplices σ and τ' are paired depends only on what simplices appear between σ and τ', not on their order. In other words, re-ordering the simplices in the critical set X_σ does not change the pairing of the (first or last) simplex in the set that is paired with σ. This argument implies that when we add a simplex to the critical set, we don't lose any of the simplices already in it.

Lemma 7. *Suppose X_σ^{k-1} is a critical set and τ_k appears immediately before or after the set (depending on the direction that τ is moving). Suppose that transposing X_σ^{k-1} and τ_k changes the pairing of σ to τ_k. Then $X_\sigma^k = X_\sigma^{k-1} \cup \tau_k$ becomes the critical set after the transposition.*

A key property of the critical set, expressed in the following lemma, is that it is resilient under transpositions. If a simplex τ' can transpose with the critical set without becoming paired with σ, then the critical set does not change after the transposition.

Lemma 8. *Suppose X_σ^{k-1} is a critical set and τ_k appears immediately before or after the set (depending on the direction that τ is moving). Suppose that transposing X_σ^{k-1} and τ_k does not change the pairing of σ. Then $X_\sigma^k = X_\sigma^{k-1}$ remains critical after the transposition.*

Proof. Suppose we are decreasing the value of τ and, therefore, by definition, any simplex in X_σ^{k-1}, when placed last, is paired with σ. Let τ' be this last simplex in X_σ^{k-1} in the filtration order. Transpose τ_k with

[2] The only possible values of q are $(p-1)$ or $(p+1)$.

all but the last simplex in the critical set. The last simplex, τ', remains paired with σ (by Corollary 5). Now transpose τ_k and τ'. Their pairing doesn't change, since τ' does not become paired with σ by the assumption of the lemma, and τ' remains paired with σ. Since this argument holds for every $\tau' \in X_\sigma^{k-1}$, the critical set does not change.

The same argument applies when increasing the value of τ by replacing "last" with "first." □

Lemmas 7 and 8 together mean that as we move simplex τ, the critical set can only grow: simplices enter, but never leave. Lemma 8 suggests Algorithm 2 for changing the value of τ, using transpositions [11]: for each of the m_p p-simplices with values between d and d', transpose it past the critical set. If its pairing changes to σ (or if the transposition is impossible because it is a face or a coface of one of the simplices in the critical set), add it to the critical set. Because each transposition takes linear time [11], the first for-loop runs in $\mathrm{O}(m^2 n)$ time. The second for-loop can be implemented as a breadth-first search through the graph of the face–coface relationships (called a Hasse diagram), so it takes $\mathrm{O}(dm)$ time, where d is the dimension of K. Because $d < n$, the former dominates, and we get $\mathrm{O}(m^2 n)$ running time for the whole algorithm.

Algorithm 2 Moving τ using individual transpositions.

```
1:  X_σ^1 = {τ}
2:  for each p-simplex τ_k with f(τ_k) from d to d' do
3:      transpose τ_k with each simplex in X_σ^{k-1},
4:          updating the pairing using the algorithm in [11]
5:      if τ_k becomes paired with σ then
6:          X_σ^k = X_σ^{k-1} ∪ {τ_k}
7:          transpose τ_k with each simplex in X_σ^{k-1},
8:              undoing the transpositions in Line 4,
9:              returning it to the opposite end of X_σ^k
10:     else
11:         X_σ^k = X_σ^{k-1}
12: for each τ ∈ X_σ^{m_p} do
13:     set f(τ) = d' (ties broken implicitly via the original order)
14:     if d' > d then
15:         // move cofaces
16:         for each simplex ρ ⊇ τ, and d < f(ρ) < d' do
17:             set f(ρ) = d'
18:     else
19:         // move faces
20:         for each simplex σ ⊆ τ, and d' < f(σ) < d do
21:             set f(σ) = d'
```

Remark. *The transpositions in Line 9 are unnecessary, but they simplify the proofs below.*

Our main contribution is an algorithm for identifying the entire critical set in $\mathrm{O}(m)$ time, without having to perform the transpositions. The resulting effect is illustrated in Figure 1, where the gradient flow implicitly traced by Algorithm 2 follows the brown curve. By identifying the critical set, we can move directly to the final destination — taking a "big step" — as illustrated with the blue curve.

3.2 Increase Death

Suppose we are trying to increase the value of τ, paired with σ, from d to d'. And suppose decomposition $D_p = R_p U_p$ is obtained using a lazy reduction. Then it suffices to examine the row $U_p[\tau, \cdot]$ to identify the

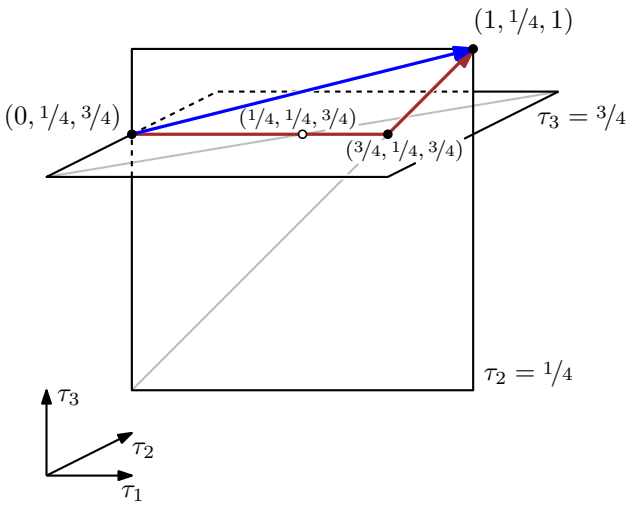

Figure 1: Three simplices, τ_1, τ_2, τ_3 have initial values $(0, 1/4, 3/4)$. Our goal is to increase the value of τ_1 to 1, and we assume that its critical set includes simplex τ_3, but not τ_2. The final simplex values are $(1, 1/4, 1)$. The path taken by the gradient flow is shown in brown. The big step that our algorithm identifies, in blue.

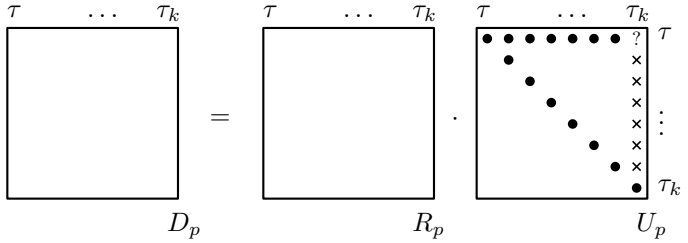

Figure 2: Subset of the matrices $D_p = R_p U_p$ involved in the proof of Theorem 9.

simplices that must move together with τ. Specifically,

$$X_\sigma = \left\{ \tau_i \;\middle|\; \begin{array}{l} d \le f(\tau_i) \le d', \\ U_p[\tau, \tau_i] \neq 0 \end{array} \right\} \tag{2}$$

is the final critical set that we would accumulate under the gradient flow. In other words, it suffices to move simplices in X_σ — and their cofaces — directly by setting $f(\tau_i) = d'$.

Theorem 9. *The critical set X_σ defined in Equation (2) is the set of simplices accumulated by Algorithm 2, when increasing the value of a negative simplex τ.*

Proof. Suppose there are m_p p-simplices with $d \le f(\tau_i) \le d'$. Denote the first k of them with Y_k. We prove the claim by induction. Restrict the set X_σ from Equation (2) to the set

$$X_\sigma^k = X_\sigma \cap Y_k. \tag{3}$$

We claim that this set is the same as its namesake in Algorithm 2.

The statement is trivially true for the base case: $X_\sigma^1 = \{\tau\}$.

Consider the steps taken by Algorithm 2. Suppose the claim is true after $k-1$ steps. By induction, all simplices τ_i in X_σ^{k-1} have $U_p[\tau, \tau_i] \neq 0$. Since the reduction is lazy, Corollary 2 implies $\sigma_i = \operatorname{low} R_p[\cdot, \tau_i] \le \sigma$. At step k, we decide whether simplex τ_k needs to be added to the critical set.

Consider the subset of the $D_p = R_p U_p$ decomposition, restricted to the critical set and τ_k, i.e., simplices in the range $\tau \ldots \tau_k$; see Figure 2. We can zero out the column $U_p[\cdot, \tau_k]$ in this range using row operations in

matrix U_p, adding multiples of row $U_p[\tau_k, \cdot]$ to the rows $U_p[\tau_i, \cdot]$ above it. The corresponding operations in matrix R_p, which maintain the decomposition, subtract multiples of columns $R_p[\cdot, \tau_i]$ from column $R_p[\cdot, \tau_k]$. Denote the former by matrix W and the latter by W^{-1}. We have $D_p = (R_p \cdot W^{-1}) \cdot (W \cdot U_p) = R'_p U'_p$.

Once column $U'_p[\cdot, \tau_k]$ is zeroed out, we can transpose τ_k with the critical set X_σ^{k-1}. The columns of the critical set may need to be reduced further, but the column $R'_p[\cdot, \tau_k]$ is already reduced, and therefore we can infer the pairing of τ_k after the transposition.

Denote by $\sigma_k = \operatorname{low} R_p[\cdot, \tau_k]$, the pair of τ_k before the transposition. If $\sigma_k > \sigma$, then it remains so after the transposition: by the inductive hypothesis $\sigma_i = \operatorname{low} R_p[\cdot, \tau_i] \leq \sigma$ for all $\tau_i \in X_\sigma^{k-1}$, and therefore adding these columns to $R_p[\cdot, \tau_k]$ doesn't change its lowest non-zero. We note that because the reduction is lazy, in this case $U_p[\tau, \tau_k] = 0$ by Lemma 1.

If $\sigma_k < \sigma$, then we need to examine $U_p[\tau, \tau_k]$. If it is zero, then after the transposition $\sigma'_k = \operatorname{low} R'_p[\cdot, \tau_k]$ remains less than σ, and therefore τ_k does not become paired with σ. If $U_p[\tau, \tau_k] \neq 0$, then $\sigma = \operatorname{low} R'_p[\cdot, \tau_k]$ and τ_k enters the critical set.

To summarize, τ_k enters the critical set X_σ^k if and only if $U_p[\tau, \tau_i] \neq 0$. In other words, X_σ^k in Equation (3) and in Algorithm 2 are the same.

It is crucial to our argument that if τ_k does not enter the critical set, and therefore moves past it, that $U_p[\tau, \tau_k] = 0$. Because of this property, the row $U_p[\tau, \cdot]$ does not change via matrix updates in the induction, and therefore the entries that we encounter in the row at any step are the same.

□

3.3 Decrease Death

Suppose we are trying to decrease the value of simplex τ from d to d'. And suppose decomposition $R_p = D_p V_p$ is obtained using a lazy reduction. Then it suffices to examine the column $V_p[\cdot, \tau]$. Specifically,

$$X_\sigma = \left\{ \tau_i \;\middle|\; \begin{array}{l} d' \leq f(\tau_i) \leq d, \\ V_p[\tau_i, \tau] \neq 0 \end{array} \right\} \tag{4}$$

is the final critical set that we would accumulate under the gradient flow. In other words, it suffices to move simplices in X_σ — and their faces — directly by setting $f(\tau_i) = d'$.

Theorem 10. *The critical set X_σ defined in Equation (4) is the set of simplices accumulated by Algorithm 2, when decreasing the value of a negative simplex τ.*

Proof. Suppose there are m simplices with $d' \leq f(\tau_i) \leq d$. Denote the last k of them with Y_k. We prove the claim by induction. Restrict the set X_σ from Equation (4) to the set

$$X_\sigma^k = X_\sigma \cap Y_k. \tag{5}$$

We claim that this set is the same as its namesake in Algorithm 2.

The statement is trivially true for the base case: $X_\sigma^1 = \{\tau\}$.

Consider the steps taken by Algorithm 2. Suppose the claim is true after $k-1$ steps. By induction, all simplices τ_i in X_σ^{k-1} have $V_p[\tau_i, \tau] \neq 0$. Since the reduction is lazy, Corollary 3 implies $\sigma_i = \operatorname{low} R_p[\cdot, \tau_i] \geq \sigma$. At step k, we decide whether simplex τ_k needs to be added to the critical set.

Consider the subset of the $R_p = D_p V_p$ decomposition, restricted to the τ_k and the critical set, i.e., simplices in the range $\tau_k \ldots \tau$; see Figure 3. Suppose we transpose τ_k with all the simplices in the critical set X_σ^{k-1}, except for the last simplex τ. Denote the updated matrices R'_p and V'_p. By Corollary 5, the pairing may change only among the transposed simplices. In particular, columns $R'_p[\cdot, \tau] = R_p[\cdot, \tau]$ and $V'_p[\cdot, \tau] = V_p[\cdot, \tau]$ do not change.

If $V_p[\tau_k, \tau] = 0$, then we can transpose τ and τ_k without changing the pairing. In particular, τ remains paired with σ. If $V_p[\tau_k, \tau] \neq 0$, then from the contrapositive of Lemma 1, before the transposition $\sigma_k = \operatorname{low} R_p[\cdot, \tau_k] > \sigma$. From the inductive assumption (that together with Lemma 1 implies that for all $\tau_i \in X_\sigma^{k-1} - \{\tau\}$, their pairs $\sigma_i > \sigma$) and from Corollary 5, after transposing τ_k to just before τ, its pair

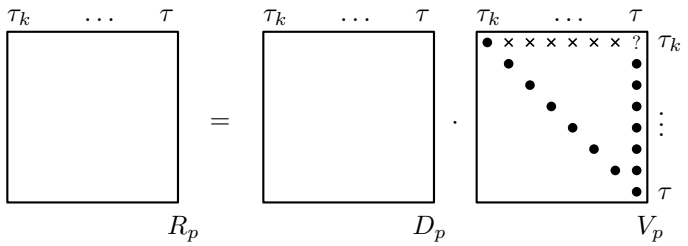

Figure 3: Subset of the matrices $R_p = D_p V_p$ involved in the proof of Theorem 10.

$\sigma'_k = \operatorname{low} R'_p[\cdot, \tau_k] > \sigma$. To perform the final transposition, we need to zero out $V'_p[\tau_k, \tau]$, which adds a multiple of column $R'_p[\cdot, \tau_k]$ to $R'_p[\cdot, \tau]$. After the transposition, we undo the operation in the column of τ_k, which becomes

$$R_p[\cdot, \tau_k] - (1/\alpha) \cdot (R_p[\cdot, \tau] - \alpha \cdot R_p[\cdot, \tau_k]) = -(1/\alpha) \cdot R_p[\cdot, \tau]$$

where $\alpha = V_p[\tau_k, \tau]$. It follows that τ_k becomes paired with σ and therefore enters the critical set.

To summarize, τ_k enters the critical set X^k_σ if and only if $V_p[\tau_k, \tau] \neq 0$. In other words, X^k_σ in Equation (5) and in Algorithm 2 are the same.

It is crucial to our argument that if τ_k does not enter the critical set, and therefore moves past it, that $V_p[\tau_k, \tau] = 0$. Because of this property, column $V_p[\cdot, \tau]$ does not change via matrix updates in the induction, and therefore the entries that we encounter in the column at any step are the same. This property, guaranteed by the use of the lazy reduction, is used in the proof via Lemma 1. □

Remark. *The proof of Theorem 10 carries through word-for-word if τ is a positive unpaired simplex. This makes it possible to decrease the birth value of points at infinity by examining the respective column in matrix V_p. Notably, the argument breaks if simplex τ is positive and paired. In this case the updates of the rows in matrix R_p complicate the transpositions. It is not difficult to construct examples of the latter, where it is not enough to examine the columns of matrix V_p.*

3.4 Increase or Decrease Birth

Thanks to duality, we are done. Increasing and decreasing death in the previous subsection really means moving p-simplex τ, with non-zero $R_p[\cdot, \tau]$, either to the left or to the right in the filtration and matrices D_p, R_p, V_p, and U_p. In the dual matrices $D^\perp_p, R^\perp_p, V^\perp_p$, and $U^\perp_p$, a simplex σ, with non-zero $R^\perp_p[\cdot, \sigma]$ is a birth simplex in a finite pair (σ, τ). Moving it to the left in the anti-transposed matrices, whose rows and columns are ordered in the reverse filtration order, translates to *increasing* its value in the filtration. Moving the simplex to the right, to *decreasing* its value.

As a result we get the following two theorems by substituting the dual matrices into the proofs of Theorems 9 and 10.

Theorem 11. *Critical set*

$$X_\tau = \left\{ \sigma_i \;\middle|\; \begin{array}{l} d \leq f(\sigma_i) \leq d', \\ V^\perp_p[\sigma_i, \sigma] \neq 0 \end{array} \right\}$$

is the set of simplices accumulated by Algorithm 2, when increasing the value of a positive $(p-1)$-simplex σ paired with τ.

Theorem 12. *Critical set*

$$X_\tau = \left\{ \sigma_i \;\middle|\; \begin{array}{l} d' \leq f(\sigma_i) \leq d, \\ U^\perp_p[\sigma, \sigma_i] \neq 0 \end{array} \right\}$$

is the set of simplices accumulated by Algorithm 2, when decreasing the value of a positive $(p-1)$-simplex σ paired with τ.

Table 1: Summary of operations and their respective rows and columns for $(p-1)$-dimensional σ and p-dimensional τ.

Operation	Row/column	Extra
Increase birth (σ) in (σ, τ)	$V_p^{\perp}[\cdot, \sigma]$	cofaces
Decrease birth (σ) in (σ, τ)	$U_p^{\perp}[\sigma, \cdot]$	faces
Increase death (τ) in (σ, τ)	$U_p\ [\tau, \cdot]$	cofaces
Decrease death (τ) in (σ, τ)	$V_p\ [\cdot, \tau]$	faces
Increase birth (σ) in (σ, ∞)	$V_p^{\perp}[\cdot, \sigma]$	cofaces
Decrease birth (σ) in (σ, ∞)	$V_p\ [\cdot, \sigma]$	faces

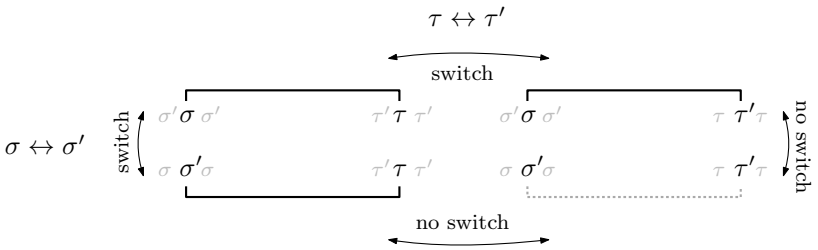

Figure 4: Top-left: simplices σ and τ are paired with each other and define the respective critical sets. $\sigma' \in X_\tau$ and $\tau' \in X_\sigma$ are contiguous with σ and τ (they come before if we are decreasing the respective value, and after, if we are increasing it). The assumption that simplices σ' and τ' are in critical sets implies that as we transpose them with σ and τ, there is a switch in the pairing (top row and left column). If after both transpositions, σ' and τ' are not paired with each other (gray dotted line in the bottom-right), then no pairing switches during the second transposition. This determines the pairing between the four simplices: σ is paired with τ' and σ' is paired with τ.

Remark. *The remark at the end of the previous subsection about examining the column $V_p[\cdot, \sigma]$ to decrease the value of an unpaired simplex σ translates to examining the column $V_p^{\perp}[\cdot, \sigma]$ to increase its value.*

Table 1 summarizes which matrices participate in each case.

3.5 Consistency of Critical Sets

Lemmas 7 and 8 imply that individual critical sets are well-defined: as we add simplices to a critical set during optimization, it can never lose a simplex. But what happens when we change birth and death simultaneously? In this case, we have to settle for an additional assumption, namely that point p defining the singleton loss has multiplicity one.

Theorem 13. *If $p = (f(\sigma), f(\tau))$ has multiplicity one, then X_τ and X_σ don't change under permutation, i.e., for every $\tau' \in X_\sigma$, if we swap it with τ, then its critical set $X_{\tau'} = X_\tau$, and for every $\sigma' \in X_\tau$, if we swap it with σ, then its critical set $X_{\sigma'} = X_\sigma$.*

Proof. Consider arbitrary simplices τ' in X_σ and σ' in X_τ. By Definition 6, if τ and τ' are swapped in the filtration, then σ is paired with τ'. Similarly, if σ and σ' are swapped, then σ' is paired with τ. Without loss of generality, we can assume that both pairs of simplices — σ and σ' as well as τ and τ' — are contiguous in the filtration. Then the two swaps above are transpositions of contiguous simplices. The assumptions about $\sigma' \in X_\tau$ and $\tau' \in X_\sigma$ imply that either transposition (top and left in Figure 4) leads to a switch in pairing. If after transpositions of both pairs, σ' and τ' are not paired with each other (lower-right part of Figure 4), then no switch in pairing occurs during the second of the two transpositions. This necessarily implies that σ' is paired with τ and σ is paired with τ'. In other words, there are two persistence pairs between the critical sets, meaning point $p = (f(\sigma), f(\tau))$ in the diagram has multiplicity greater than one. □

Remark. *The theorem applies to every critical set during the optimization. The point defining singleton loss may start out having multiplicity one, but gain higher multiplicity as the critical sets grow.*

3.6 Faces and Cofaces

After identifying the critical set X_σ, we need to move all the cofaces (when increasing) or faces (when decreasing) of every simplex in the set, to ensure that the new simplex values define a filtration. In Algorithm 2, the for-loop in Lines 12 to 21 performs the required update. For a general filtration, we simply execute the same for-loop, which takes $\mathrm{O}(dm)$ time. Because we can identify the critical set in $\mathrm{O}(m)$ time, finding the faces and cofaces dominates the running time. However, since d is a small constant in all practical applications, linearity in m is most important.

In practice, an explicit update of faces and cofaces is unnecessary. The function $f : K \to \mathbb{R}$ that defines the filtration is derived from the input data. The gradients on the simplices are backpropagated through f to the input values, which are updated by the optimization. When an updated function $f' : K \to \mathbb{R}$ is derived from the updated data, it satisfies the face condition and defines a filtration by construction.

For example, consider a lower-star filtration (which we use to compute persistence of scalar fields in all our experiments in the next section): given a function, $\hat{f} : \operatorname{Vert} K \to \mathbb{R}$, on the vertices of the simplicial complex, we extend it to all the simplices, $f(\sigma) = \max\{\hat{f}(v) \mid v \in \sigma\}$. When we get a gradient $\partial\mathcal{L}/\partial f(\sigma)$, which we backpropagate to $\partial\mathcal{L}/\partial\hat{f}(v)$, where $v = \arg\max_{v' \in \sigma} \hat{f}(v')$. After taking a step, following this gradient, we get a new function on the vertices $\hat{f}'$. Because the new filtration is constructed as a lower-star filtration of this function, we are guaranteed that all the faces precede σ and all of its cofaces come after. The same argument applies to the Vietoris–Rips filtration, Čech filtration, alpha filtration, etc. In all such cases, the $\mathrm{O}(dm)$ term is eliminated from the running time, leaving only $\mathrm{O}(m)$.

We note that it may still be worthwhile to compute and move the faces or cofaces explicitly. The $\mathrm{O}(dm)$ overhead is minor, but more gradient information gets propagated to the input data.

3.7 Combined Loss

Given a general loss, $\mathcal{L} = \sum_{(p,q)\in M}(p-q)^2$, defined by an arbitrary matching M, typically recomputed after every step of the optimization, we can compute the target values for each simplex prescribed by the singleton losses defined by the individual terms of the sum. For a simplex σ, with the initial value $f(\sigma) = a$, we get a set of target values $\{a_1, a_2, \ldots\}$, one for each singleton loss. (If a singleton loss doesn't prescribe a value to a simplex, then the corresponding value a_i is missing from the target set, which can be empty as a result.) Ultimately, we want to define a gradient on the individual simplices that would allow us to take (small) optimization steps, but to do so we need to decide on one target value for each simplex.

There are several ways to combine the target values into one. We choose to set

$$a' = a_i, \quad \text{where } i = \arg\max_j\{|a - a_j|\}$$

as the target value for σ, i.e., moving it as far as possible in the filtration. (Two other strategies are considered in Appendix B.) This is a heuristic, without a strong justification, but with the following reasoning behind it. When all simplices of a given dimension are moving in the same direction (e.g., all 1-simplices increase and all 2-simplices decrease their values), most simplices get prescribed values a_i that are lower bounds on how far they need to move to solve the singleton loss. Put another way, all but the first or the last simplex in the critical set can move farther than their stated target. So taking the maximum is a way to satisfy all lower bounds simultaneously. Another reason for the maximum is that for the simplification loss $\mathcal{L}_\varepsilon$, defined in Equation (1) at the beginning of Section 3, when applied to diagrams of dimension 0 or codimension 1, maximum gives the optimal solution in one step. Intuitively, the reason is that when multiple values are prescribed to the same simplex, it means that it belongs to multiple nested topological features. (A formal proof of this claim requires a lot of new machinery, which is why we omit it. The claim is only a minor motivation for our heuristic choice.)

Algorithm 3 summarizes our overall method.

Algorithm 3 Critical set method.

Input: $\mathcal{L} = \sum_{(p_i,q_i)\in M}(p_i - q_i)^2$
for *each* $(p_i, q_i) \in M$ **do**
  let $p_i = (b_i, d_i) = (f(\sigma_i), f(\tau_i))$; $q_i = (b'_i, d'_i)$
  $X_b = \left\{ \sigma_j \;\middle|\; \begin{array}{l} V^{\perp}[\sigma_j, \sigma_i] \neq 0 \text{ and } b_i \leq f(\sigma_j) \leq b'_i; \text{ or} \\ U^{\perp}[\sigma_i, \sigma_j] \neq 0 \text{ and } b'_i \leq f(\sigma_j) \leq b_i \end{array} \right\}$
  $X_d = \left\{ \tau_j \;\middle|\; \begin{array}{l} U[\tau_i, \tau_j] \neq 0 \text{ and } d_i \leq f(\tau_j) \leq d'_i; \text{ or} \\ V[\tau_j, \tau_i] \neq 0 \text{ and } d'_i \leq f(\tau_j) \leq d_i \end{array} \right\}$
  // omitted: find faces/cofaces if necessary
  for $\sigma_j \in X_b$ **do**
    append b'_i to target$[\sigma_j]$
  for $\tau_j \in X_d$ **do**
    append d'_i to target$[\tau_j]$
for *each* σ **do**
  if target$[\sigma]$ *is empty* **then**
    $f'(\sigma) = f(\sigma)$
  else
    $j = \arg\max_j \{|f(\sigma) - \text{target}[\sigma][j]|\}$
    $f'(\sigma) = \text{target}[\sigma][j]$
return $\forall \sigma, \partial\mathcal{L}/\partial f(\sigma) = 2(f(\sigma) - f'(\sigma))$

Remark. *In practice, one typically uses an automatic differentiation library. Instead of computing the gradients explicitly, as in Algorithm 3, one computes the corresponding loss* $\mathcal{L} = \sum(f(\sigma) - f'(\sigma))^2$*, where the summation is over all simplices* σ *with non-empty* target$[\sigma]$*. Value* $f(\sigma)$ *must be automatically differentiable and* $f'(\sigma)$ *is a constant. Backpropagation takes care of evaluating* $\nabla\mathcal{L}$ *with respect to* $f(\sigma)$ *and any variables on which* $f(\sigma)$ *depends.*

Decreasing loss. In general, the heuristic of taking the maximum displacement as the target value is not guaranteed to decrease the loss locally. For such a guarantee, we need to assume that all the simplex values are distinct and take a sufficiently small step in any direction whose individual components have the same sign as the negative of the gradient of the loss. (The components on which the gradient of the loss is zero can have any sign.) This follows from the loss being additive: each individual term $(p_i - q_i)^2$ decreases if the coordinates of p_i are brought closer to q_i.

If one wanted to ensure that the loss decreases locally, it is easy to enforce this condition explicitly by fixing the gradient values of the critical simplices to be the same as the gradient given by the loss. This results in an alternative heuristic for combining singleton losses. We present one such example under the name `fca` in Appendix B.

Another setting where the loss is guaranteed to decrease using Algorithm 3 directly is if all points in the persistence diagram are prescribed the same kind of movement, i.e., their birth and death values either increase or decrease in tandem.

4 Experiments

In this section, we compare optimization that computes gradients by identifying critical sets of singleton losses, as explained in the previous section, to the existing approach in the literature that defines the gradients on the pairs of simplices that define the persistence pairing, as explained in the Introduction. Below, in figure legends, we refer to our new method as "Critical set" and to the previous method as "Diagram."

Vineyards. In all our experiments we get a series of diagrams D_t indexed by the optimization step. We visualize two of their projections to understand their evolution.

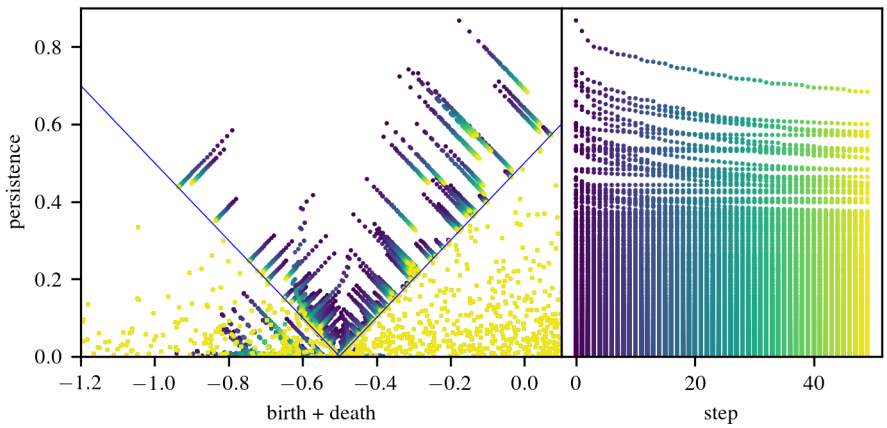

Figure 5: Vineyard of the optimization guided by the simplification of a sublevel set in a 0-dimensional diagram of the magnetic reconnection dataset, using the **diagram method**. Learning rate is 0.2, without momentum. The color encodes the time step. The left projection makes it clear that many of the points do not reach the quadrant boundary after 50 steps.

Data. We use two scientific datasets from the "Open Scientific Visualization Datasets" collection [21].

- Rotstrat [27]: temperature field of a numerical simulation of rotating stratified turbulence.
- Magnetic reconnection [19]: a single time step from a computational simulation of magnetic reconnection. This dataset has a visible geometric structure that looks like a curved tunnel in the middle.

We downsampled the data (still keeping it larger than most data sets used for topological optimization in the literature). All our experiments are done on datasets of size 32^3.

We use upper- or lower-star filtrations, described in Section 3.6, to compute persistence of the data. Because both use max or min to assign values to the simplices, we apply the same maximum displacement construction as in Section 3.7 to the vertices: if the same vertex is prescribed different gradients by different simplices, we keep the one that results in the largest displacement.

4.1 Sublevel Set Simplification

This experiment is motivated by the simplification of the decision boundary of a neural network [10], formulated as a level set. The authors phrase their loss in terms of well groups [17, 6]. For simplicity (to avoid having to introduce new constructions), we do not simplify the level set, but rather a sublevel set. Given a function $f : \mathbb{X} \to \mathbb{R}$, denote with $\mathbb{X}_a = f^{-1}(\infty, a]$ its sublevel set. A topological feature exists in this sublevel set, if its birth value is less than a and its death value is greater than a. Geometrically, we want to eliminate the points of the persistence diagram that lie in the quadrant defined by $b \leq a$ and $d \geq a$. We match each such point (b_i, d_i) to the closest point on the boundary of the quadrant, i.e., either (b_i, a) or (a, d_i).

We ran this experiment for the magnetic reconnection dataset. The threshold was chosen in such a way that the quadrant contains a large portion of the points.

Figures 5 and 6 present the vineyards of the two optimization procedures. The color of the point encodes the step number, in both projections. The blue lines show the quadrant that we want to make free of the diagram points.

Because topological features are intertwined in complicated ways, it is impossible to move only the points in the quadrant. The points outside of the quadrant are moving too, and some of the points in the quadrant are not moving directly to their prescribed target. This is expected in both cases. What is notable is that using our critical set method, the points move much more efficiently: after 50 steps, all points end up on the boundary of the quadrant, when using the critical set method, but many do not reach the boundary, when using the diagram method.

To better compare the two optimization methods, we plot the value of the diagram loss at each step of

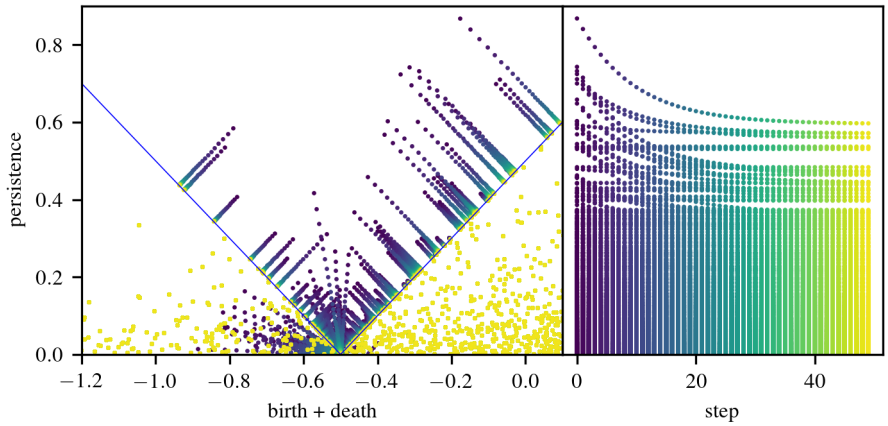

Figure 6: Vineyard of the optimization guided by the simplification of a sublevel set in a 0-dimensional diagram of the magnetic reconnection dataset, using the **critical set method**. Learning rate is 0.2, without momentum. The color encodes the time step. The left projection makes it clear that all the points rapidly reach the quadrant boundary.

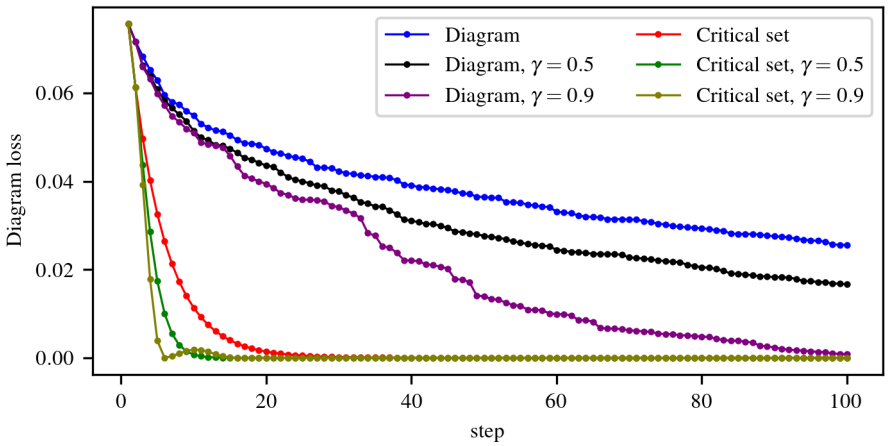

Figure 7: Comparison of the diagram losses during the optimization using the two methods. Diagram method greatly benefits from momentum. Critical set also benefits from momentum, but performs well even without it.

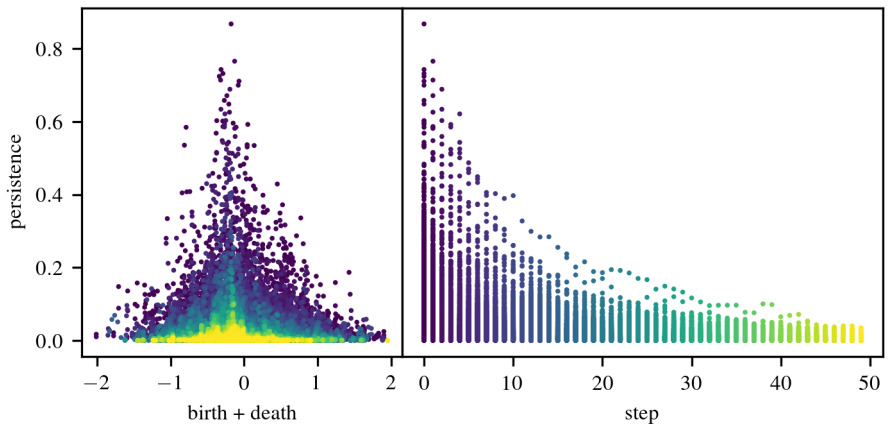

Figure 8: Vineyard of the optimization guided by the simplification loss in a 1-dimensional diagram of the Rotstrat dataset, using the **diagram method**. The color encodes the time step. $\varepsilon = \infty$, learning rate 0.2, with momentum $\gamma = 0.9$. The plot makes it clear that the points don't reach their targets after 50 steps.

the optimization in Figure 7. We used three optimization variants: standard gradient descent and gradient descent with momentum, with damping parameter $\gamma = 0.5, 0.9$. The smaller value of γ makes the influence of the gradient from the previous steps weaker. Unsurprisingly, momentum makes a big difference for the diagram method: since it needs to move large portions of the domain, but it has gradient information only on the critical simplices, the ability to keep moving simplices for several steps is crucial. Our method also benefits from momentum, but less so, and it performs well with a lower value of the damping parameter, $\gamma = 0.5$. The diagram method works best with the higher $\gamma = 0.9$, but even with this value it is not nearly as fast the critical set method, which rapidly drops to 0 with or without momentum.

4.2 Persistence-sensitive Simplification

Simplification loss was defined in Equation (1) at the beginning of Section 3: it matches all the points with persistence below a prescribed threshold ε to the diagonal.

We simplify the 1-dimensional diagrams, which is the case inaccessible to the existing combinatorial methods. The advantage of the critical set method is evident from the vineyards shown in Figures 8 and 9. The diagram method produces long trajectories of points moving towards the diagonal. The critical set method moves the points much faster, which is especially clear when comparing the right projections in the two figures.

The diagram loss plots are in Figure 10. The y-axis is logarithmic, which emphasizes the advantage of the momentum damping parameter of $\gamma = 0.5$ for the critical set method. Somewhat unexpectedly, a high value of momentum parameter ($\gamma = 0.9$) almost completely wipes out the advantage of the critical set method. For the diagram method, the momentum serves as a surrogate for the critical set: it helps to further push the points, which stopped being critical after one step.

4.3 Timing and Convergence Rate

The major downside of our method is that it requires considerably more computation per step. We must compute not only the reduced boundary matrix R, needed to read off the persistence diagram, but also matrices U and V. Moreover, since we use both homology and cohomology, we have to do this computation twice.

For example, for magnetic reconnection dataset it takes 3.4× longer to compute matrices U and V than matrix R by itself. To compute all four matrices $U, V, U^\perp$, and $V^\perp$ takes 4.2× longer than just matrix R. Because it requires an order of magnitude fewer steps — and the fraction gets smaller as the data gets larger,

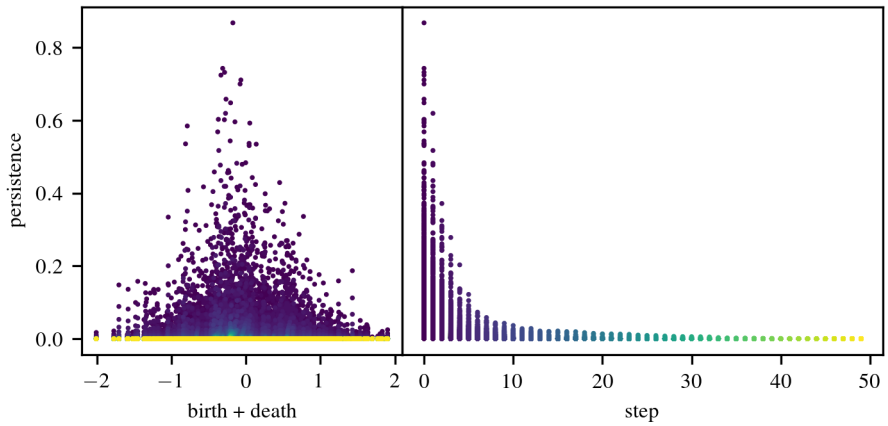

Figure 9: Vineyard of the optimization guided by the simplification loss in a 1-dimensional diagram of the Rotstrat dataset, using the **critical set method**. The color encodes the time step. $\varepsilon = \infty$, learning rate 0.2, without momentum. The points approach the diagonal much faster than in Figure 8.

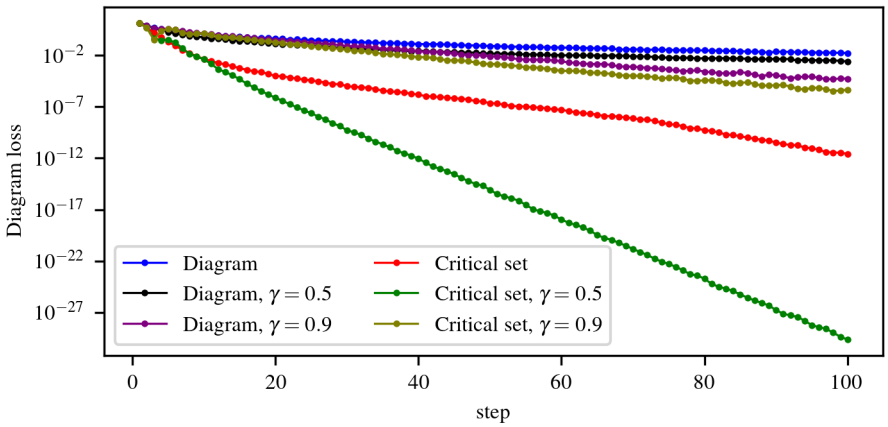

Figure 10: Comparison of the diagram losses during optimization of the simplification loss on Rotstrat dataset. Diagram methods benefits from momentum. So does critical set (for $\gamma = 0.5$), but it performs much better than the diagram method, even without it.

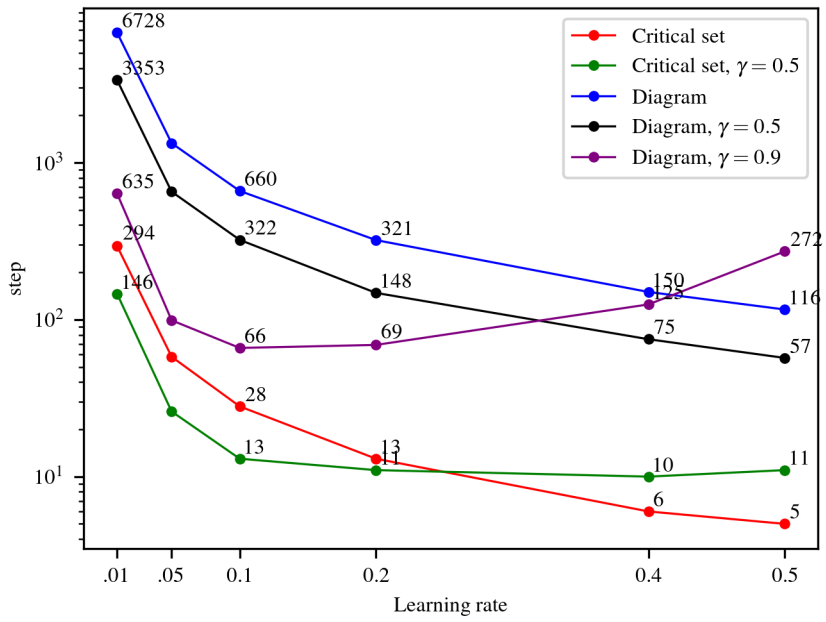

Figure 11: Comparison of the number of steps required to bring diagram loss from 7.2 to 0.001, using simplification loss on the 1-dimensional diagram of the Rotstrat dataset. $\varepsilon = \infty$.

see Appendix C — our method is still faster overall, but the result seems discouraging: much of the savings suggested by the rapidly decreasing losses are lost because of the more expensive per-step computation. We point out a possible solution in the conclusion, but meanwhile note that some flexibility exists in the formulation of the loss itself. For example, for the simplification loss, as we defined it, we move every points (b, d) to $((b + d)/2, (b + d)/2)$, which requires both to increase birth and decrease death. As summarized in Table 1, the former requires computing matrix $V^{\perp}$; the latter, matrix V. But we could also simplify the diagram by moving each point to the point (b, b) on the diagonal. This would require only decreasing the death values, and thus obviate the need to compute cohomology.

Learning rate and momentum. To study the effect of the learning rate and momentum, we simplify the Rotstrat dataset diagram in dimension 1 for different values of the hyper-parameters. For $\varepsilon = \infty$, the original value of the diagram loss is 7.2. For different values of the learning rate, we record the number of steps needed to minimize it below 0.001 using the two methods. The results are in Figure 11.

Without momentum, the critical set method has a prominent advantage for all learning rates; it requires $22 - 25\times$ fewer steps. With momentum, the diagram methods performs better. However, for large learning rates the performance of the best value of $\gamma = 0.9$ becomes worse: the corresponding purple line shoots up. No choice of the hyper-parameters is a clear winner, but if we pick the two that behave most reasonably — $\gamma = 0.5$ for the diagram method, and no momentum for the critical set method — we see about $11\times$ fewer steps for the latter.

Taking into account the computational overhead, we conclude that the overall running time of our approach is normally not worse than the diagram loss optimization, and for larger learning rates it is consistently better.

5 Conclusion

We have presented a method to accelerate optimization guided by a topological loss, formulated as a matching. The method relies on examining the cycles, chains, and related information calculated as a by-product of

persistence computation. We have shown empirically that our method reduces the number of steps required to achieve a given loss by an order of magnitude.

Warm starts. The timing results seem discouraging: a $10\times$ reduction in the number of optimization steps, combined with a $4\times$ slow-down per step caused by the computation of matrices V, U and $V^{\perp}, U^{\perp}$ results in a very modest speed-up. The fact that cohomology is not always needed provides little solace. This may seem fatal to our approach, but the recent work of Luo and Nelson [23] offers hope. Motivated by optimization, among other problems, they present a simple algorithm to quickly compute persistence pairing, given a reduction of a nearby filtration. They show that such "warm starts" significantly improve the computation speed, compared to recomputing the pairing from scratch. Crucially for us, their algorithm relies on computing the $R = DV$ decomposition. In other words, following their method, there is no extra penalty for computing matrix V, when iteratively updating persistence pairing. Working out the technical details of such a combined approach is one the most productive directions for future work.

Clearing optimization. Modern state-of-the-art implementations of persistence [3, 4], use clearing optimization [9], which identifies zero columns of matrix R, corresponding to the births of finite pairs, without reducing them explicitly. Such columns are not needed when we move points closer to the diagonal — increasing birth or decreasing death — but the absence of the corresponding operations in matrix U presents a problem, when we want to increase death or decrease birth. Although few of the losses proposed in the literature need such operations, working out a complete method for combining our construction with the clearing optimization is another worthwhile direction for future work.

Combined losses. Given a matching, we combine multiple singleton losses by taking the maximum displacement prescribed to individual simplices. This is a heuristic, without a strong justification, other than what's stated in Section 3.7. There are other natural heuristics: for example, sending each simplex to the average of its target values. We discuss two of them in Appendix B. Better understanding the resulting dynamics and finding principled ways to combine multiple singleton losses is the main theoretical question left open by our work.

In Appendix D, we compare the performance of the diagram and critical set methods on an optimization problem from [26] that back-propagates the loss past simplex values to a functional correspondence, which serves as the parameter for optimization.

Convergence guarantees. Another major direction for future work is understanding convergence guarantees of the critical set method. Carrière et al. [8] show that persistence-based losses satisfy the assumptions required by the work of Davis et al. [12], and therefore their results on the convergence of sub-gradient descent apply. Unfortunately, the gradient prescribed by the critical set method does not lie in the sub-gradient of the loss. Its entire point is to provide information on the non-critical simplices, on which the sub-gradient of the original loss is necessarily zero.

Momentum and optimization. A striking result of our experiments is that momentum often hurts critical set method, while it almost always helps the diagram method. Our general intuition is that applied to the diagram method momentum accumulates something like the critical set over the iterations of the optimization. With the critical set method, the right collection of simplices is identified by the algorithm itself and so bringing information from prior iterations just obstructs progress. Understanding the interaction of the critical set method with momentum, and optimization more broadly, is another important research topic.

Experiments with other optimizers, namely RMSProp and Adam in Appendix A, reinforce that the interaction of momentum and the critical set method is complicated and deserves future research.

Data Availability Statement

The datasets analyzed are available in the 'Open Scientific Visualization Datasets' repository by P. Klacansky, klacansky.com/open-scivis-datasets.

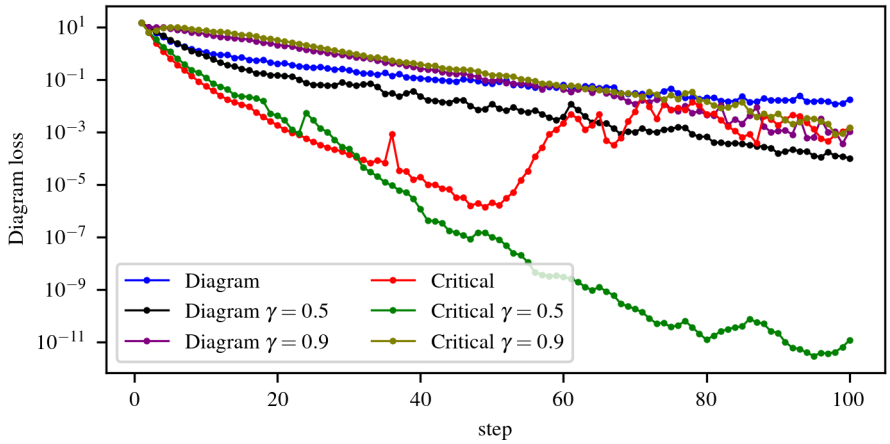

Figure 12: Simplification results for RMSProp on Rotstrat dataset.

Acknowledgments

This work was initiated under Laboratory Directed Research and Development (LDRD) funding from Berkeley Lab, provided by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. It has since been supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program and Mathematical Multifaceted Integrated Capability Centers (MMICCs) program, under Contract No. DE-AC02-05CH11231 at Lawrence Berkeley National Laboratory.

Appendices

A Different optimizers

We also tried Adam and RMSProp optimizers. The results for RMSProp are shown in Figure 12. While there are some parameters which make direct diagram loss optimization better than the critical set method, the best results are still achieved when using the critical set. The results for Adam in Figure 13 also demonstrate the efficiency of the critical set method.

B Other conflict strategies

We say that simplex σ is in conflict, if it belongs to multiple critical sets prescribed by the matching. There are different ways to resolve such conflicts. In Section 3.7, we chose to take v_k that maximizes $|f(\sigma) - v_i|$, i.e., of all the values we take the farthest from the current one. We abbreviate this choice as `max`. Averaging is another natural choice: if σ appears in the critical sets $X_{\sigma_1}, \ldots, X_{\sigma_k}$, prescribing values $v_1, \ldots, v_k$, then we assign $\frac{1}{k}\sum_{i=1}^{k} v_k$ as target value of σ. We abbreviate this choice as `avg`.

Another option is to take the average everywhere except the critical simplices. Specifically, if $\sigma = \sigma_j$ is a simplex responsible for a point that appears in the matching, then we assign v_j as the target value. Otherwise we take the average. We abbreviate this method as `fca` (Fix Critical simplices and take Average on others); its pseudocode is in Algorithm 4. This strategy imitates the gradient of the matching loss $\mathcal{L}$. Specifically, `fca` guarantees that for every critical simplex σ_j whose point in the persistence diagram appears in the matching, $\frac{\partial \mathcal{L}}{\partial \sigma_j}$ and the j-th component of our gradient are the same. If we assume general position, then the gradient of the diagram loss is 0 in all other components (infinitesimal perturbation of other simplices does

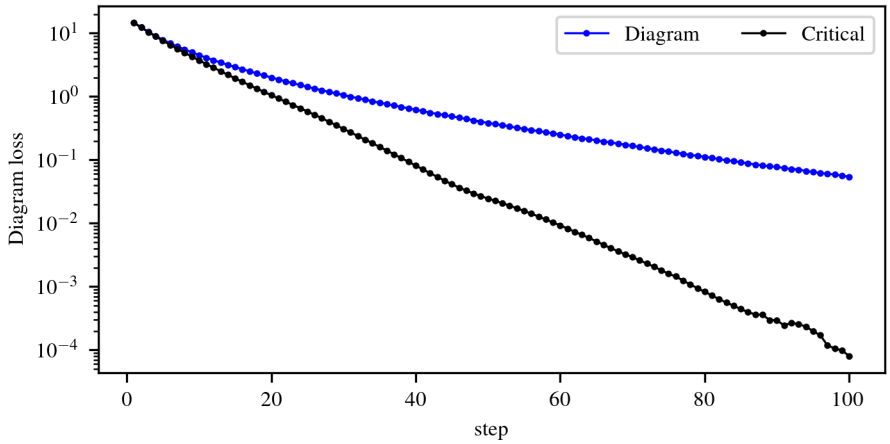

Figure 13: Simplification results for Adam on Rotstrat dataset. The parameters $\beta_1 = 0.9$ and $\beta_2 = 0.99$ are the default ones in PyTorch.

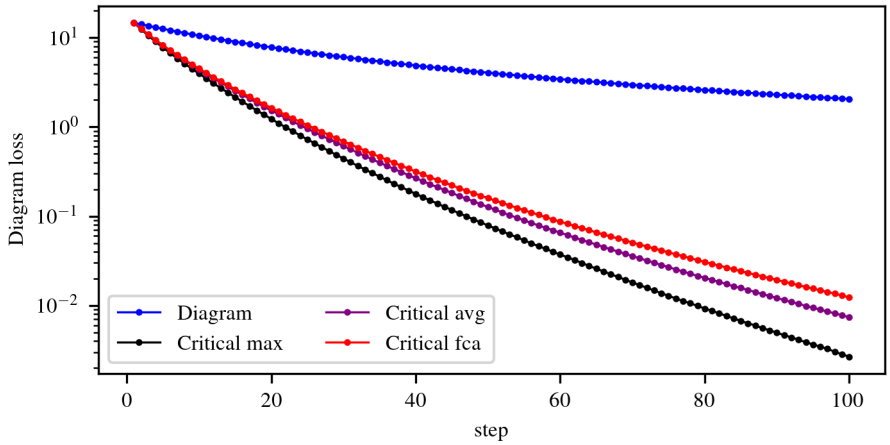

Figure 14: Comparison of different ways to combine singleton losses during optimization of the simplification loss on Rotstrat dataset, no momentum.

not change $\mathcal{L}$). In such general position, the loss $\mathcal{L}$ is guaranteed to decrease in the direction prescribed by the `fca` method, since the loss ignores values of non-critical simplices.

We ran the Rotstrat example (simplification of 1-dimensional diagram) with the same parameters as in Figure 10, varying the conflict strategy. Figures 14 and 15 show that there is little difference between the three choices. If γ is small, then taking the maximum performs best. For $\gamma = 0.9$, taking the average is slightly better, see Figure 16. In all cases, the critical set method clearly outperforms naive optimization of the diagram loss.

C Scaling experiments

Our method aims to move together all the simplices whose critical values must be modified, while the diagram method touches only the critical simplices. Thus it is reasonable to expect our method to perform better on larger inputs. Suppose we want to simplify the diagram, and we have a version of the same scalar field in different resolutions. For higher resolutions, there will be more elements in the critical set of each point, while the diagram loss identifies only one of these elements at each step.

We took the Rotstrat example and downsampled it to 3 different sizes, 32^3, 64^3 and 128^3. Then we ran

Algorithm 4 Fix Critical, Average on others.

Input: $\mathcal{L} = \sum_{(p_i,q_i)\in M}(p_i - q_i)^2$
for *each* $(p_i, q_i) \in M$ **do**
  let $p_i = (b_i, d_i) = (f(\sigma_i), f(\tau_i))$; $q_i = (b'_i, d'_i)$
  $X_b = \left\{ \sigma_j \;\middle|\; \begin{array}{l} V^{\perp}[\sigma_j, \sigma_i] \neq 0 \text{ and } b_i \leq f(\sigma_j) \leq b'_i; \text{ or} \\ U^{\perp}[\sigma_i, \sigma_j] \neq 0 \text{ and } b'_i \leq f(\sigma_j) \leq b_i \end{array} \right\}$
  $X_d = \left\{ \tau_j \;\middle|\; \begin{array}{l} U[\tau_i, \tau_j] \neq 0 \text{ and } d_i \leq f(\tau_j) \leq d'_i; \text{ or} \\ V[\tau_j, \tau_i] \neq 0 \text{ and } d'_i \leq f(\tau_j) \leq d_i \end{array} \right\}$
  // omitted: find faces/cofaces if necessary
  for $\sigma_j \in X_b$ **do**
    append b'_i to target$[\sigma_j]$
  for $\tau_j \in X_d$ **do**
    append d'_i to target$[\tau_j]$
for *each* σ **do**
  if target$[\sigma]$ *is empty* **then**
    $f'(\sigma) = f(\sigma)$
  else if $\sigma = \sigma_i$ for some i (which is unique) **then**
    $f'(\sigma) = b'_i$
  else if $\sigma = \tau_i$ for some i (which is unique) **then**
    $f'(\sigma) = d'_i$
  else
    $f'(\sigma) =$ average of target$[\sigma]$
return $\forall \sigma, \partial\mathcal{L}/\partial f(\sigma) = 2(f(\sigma) - f'(\sigma))$

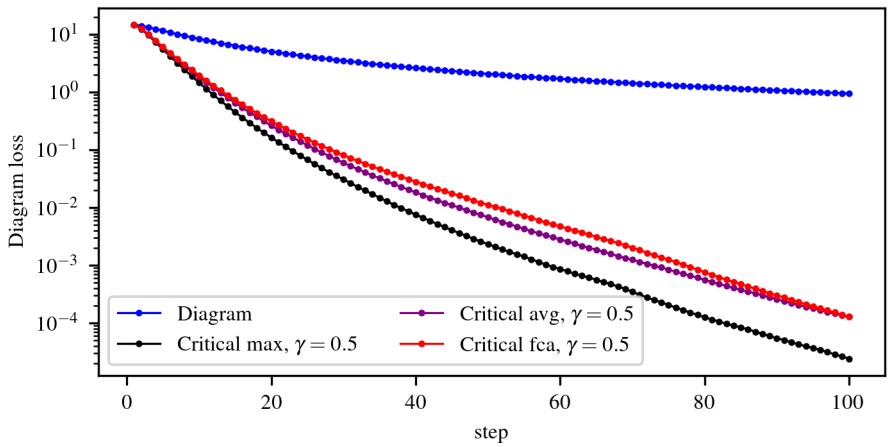

Figure 15: Comparison of different ways to combine singleton losses during optimization of the simplification loss on Rotstrat dataset, with momentum 0.5.

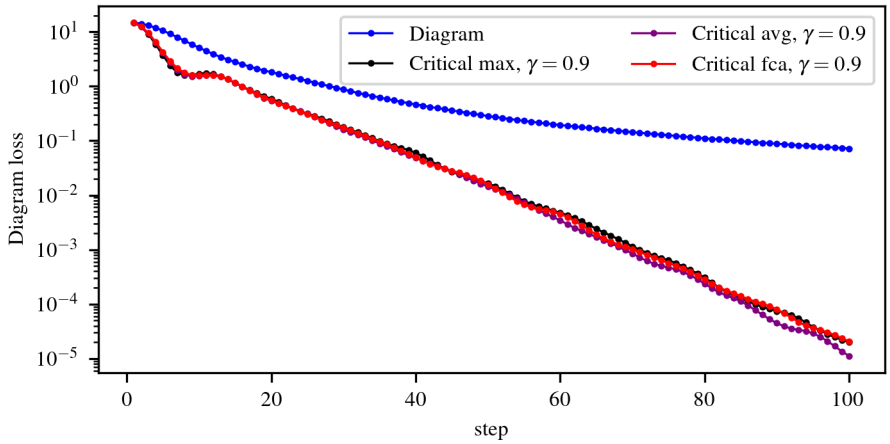

Figure 16: Comparison of different ways to combine singleton losses during optimization of the simplification loss on Rotstrat dataset, with momentum 0.9.

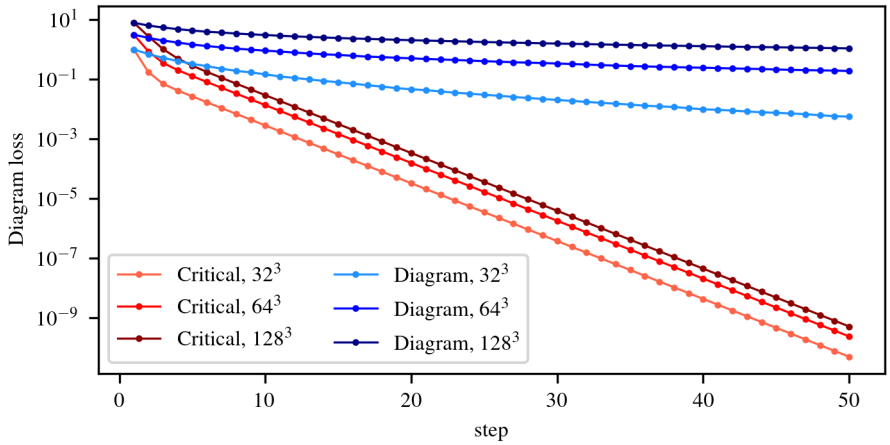

Figure 17: Comparison of the diagram losses during the optimization using the two methods on the inputs of different size. The advantage of the critical set method becomes clearer for larger inputs. There is no momentum. Learning rate is 0.1

the well group simplification of 1-dimensional diagram with the same parameters. The plots of the losses are in Figures 17 and 18. Figure 18 in particular shows that even when we use momentum with the diagram loss, the critical set method drives the diagram loss to zero significantly faster for larger inputs. We also plot the ratio of the diagram loss values in Figure 19. From this figure, we see that by step 50, the diagram loss for the 32^3 input was roughly 10^4 times smaller when optimized with the critical set method; for the 128^3 input it was 10^9 times smaller.

D Experiments with Heat Kernel Signature

We replicate some of the experiments from [26], both directly optimizing the values on a mesh and back-propagating to optimize a functional correspondence between two meshes.

Direct optimization. We performed an experiment similar to [26]. We picked a mesh from the SCAPE dataset [1] and computed HKS signature on it, using [30]. We chose $t = 0.2$ and 40 eigenvectors. Then we performed topological simplification, choosing ε to preserve the three most persistent points in the zeroth

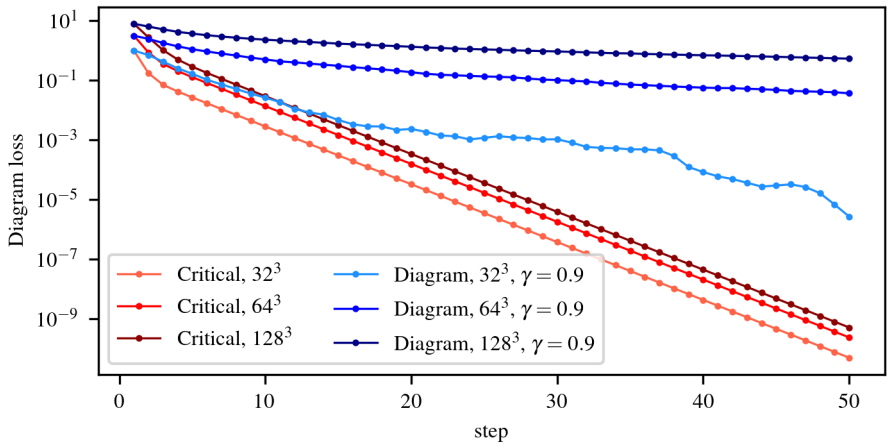

Figure 18: Comparison of the diagram losses during the optimization using the two methods on the inputs of different size. The advantage of the critical set method becomes clearer for larger inputs. Optimization is with momentum for the diagram method, $\gamma = 0.9$. Learning rate is 0.1

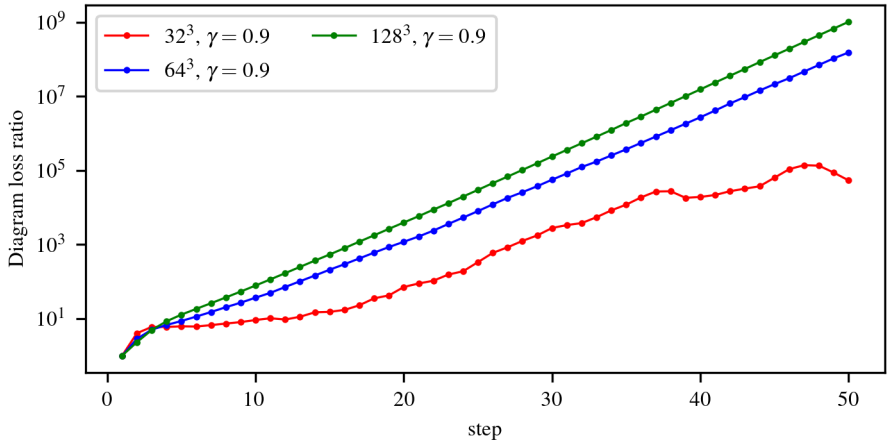

Figure 19: Ratio of the diagram losses during the optimization using the two methods on the inputs of different size: y-axis is the value of the diagram loss when optimized with the diagram method divided by the value of the diagram loss when optimized with the critical set method. Optimization is with momentum for the diagram method, $\gamma = 0.9$. Learning rate is 0.1.

(a) Original HKS function. (b) Simplified (critical set). (c) Simplified (diagram).

Figure 20: Visualization of the Heat Kernel Signature and its simplification.

diagram. The vineyards and diagrams are shown in Figures 21 and 22. Here we find that the best performance of the diagram method was for a smaller value of $\gamma = 0.5$, with $\gamma = 0.9$ the optimization diverges and starts moving points away from the diagonal, as we can see in Figure 23. This highlights a disadvantage of the diagram loss: to perform well, one needs to tune optimization parameters. The critical set method with plain gradient descent quickly drives the loss to 0, while the diagram method does not achieve the same result even after 50 steps, see Figure 24.

Figure 20 shows the results of the simplification. It is hard to tell the difference visually; one can see that both methods remove topological features by creating similar paths.

The diagram method takes 8.14 seconds. 50 steps of the critical set method take about 21.57 seconds. However, a fair comparison would be to run the critical set method until it drives the loss below the value achieved by the diagram method, which happens after 7 steps, after only 2.96 seconds.

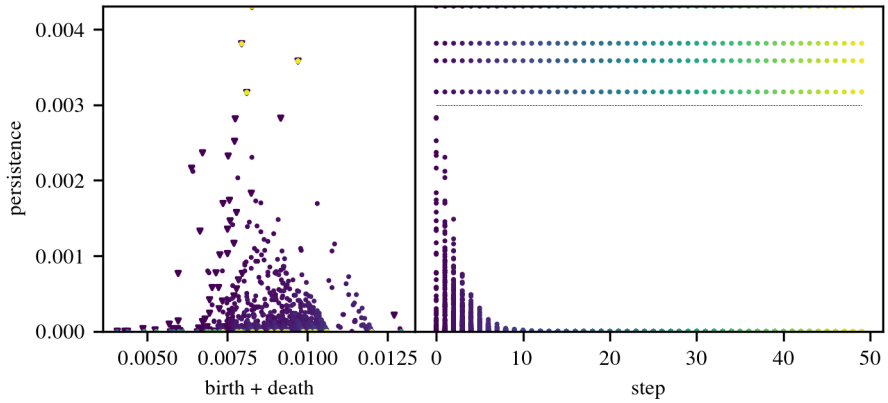

Figure 21: Vineyard of the optimization guided by the simplification of a superlevel set in a 0-dimensional diagram of the HKS, using the **critical set method**. Learning rate is 0.2, without momentum. The color encodes the time step.

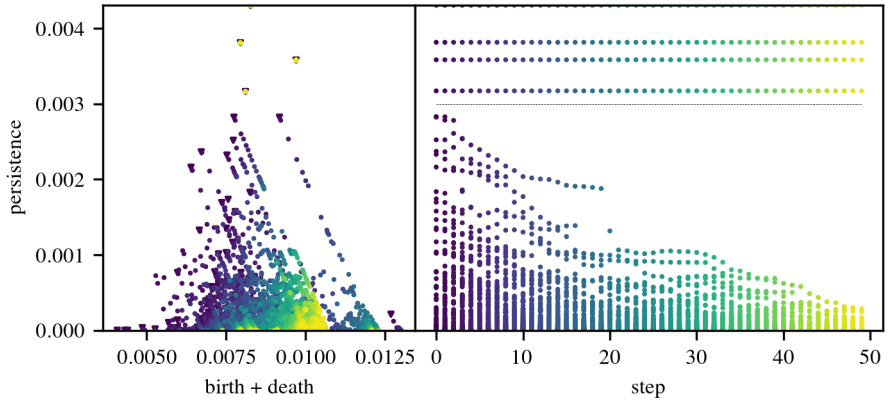

Figure 22: Vineyard of the optimization guided by the simplification of a superlevel set in a 0-dimensional diagram of the HKS, using the **diagram method**. Learning rate is 0.2, momentum $\gamma = 0.5$. The color encodes the time step.

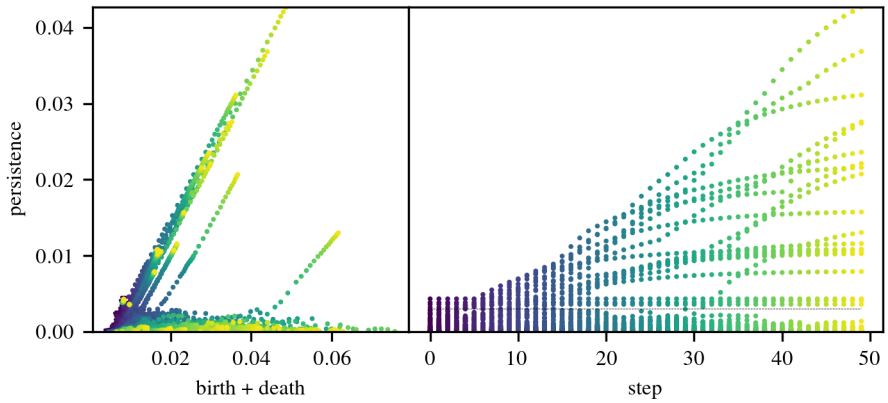

Figure 23: Vineyard of the optimization guided by the simplification of a superlevel set in a 0-dimensional diagram of the HKS, using the **diagram method**. Learning rate is 0.2, momentum $\gamma = 0.9$. The color encodes the time step. Note that the most persistent points that we wanted to preserve are no longer fixed.

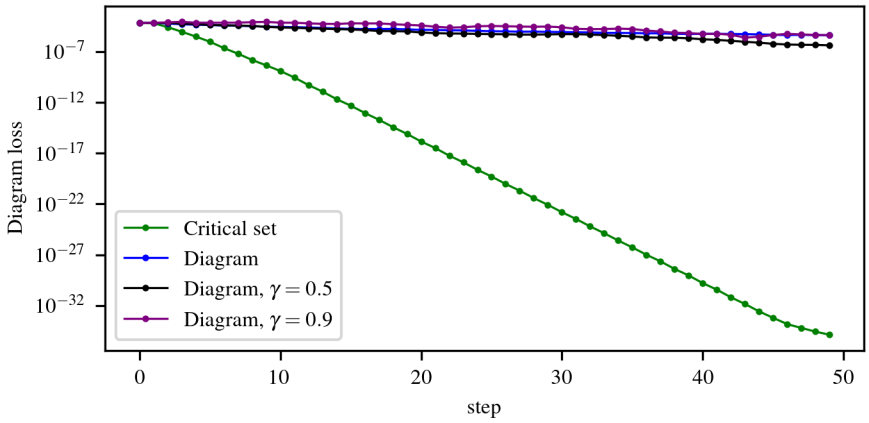

Figure 24: Comparison of the diagram losses during simplification of the HKS function. Diagram methods benefits from momentum, but the critical set significantly outperforms it.

Functional map regularization. Let us recall the methodology of the functional map correspondence method, following the notation used in [26]. We are given two manifolds (triangular meshes) $\mathcal{M}$ and $\mathcal{N}$.

First, we choose a set of basis functions, $k_{\mathcal{M}}$ and $k_{\mathcal{N}}$, on each manifold. The basis functions are the eigenfunctions of the corresponding Laplace–Beltrami operator, $L_{\mathcal{M}}$ or $L_{\mathcal{N}}$. Then we compute k_d descriptors on each manifold and expand them in the corresponding basis. We stack the column vectors with the coordinates of each descriptor into two matrices, $\mathbf{A}$ and $\mathbf{B}$, of size $k_{\mathcal{M}} \times k_d$ and $k_{\mathcal{N}} \times k_d$.

More precisely, the chosen basis functions do not span the whole space of functions on the manifold, unless we decide to use all of the eigenfunctions, because only in this case we have as many functions in the basis as vertices. Accordingly, by expanding a function in the basis we actually mean expanding its orthogonal projection on the subspace spanned by the first eigenfunctions.

The idea of a functional map correspondence is that instead of searching for a point-to-point correspondence $\mathcal{M} \to \mathcal{N}$, we search for a linear mapping from the space of all real-valued functions on $\mathcal{M}$ into the space of all functions on $\mathcal{N}$. Since we fix the bases, such a map is encoded by a matrix $\mathbf{C}$ of size $k_{\mathcal{N}} \times k_{\mathcal{M}}$. There are two reasonable requirements to impose on $\mathbf{C}$: 1) if the descriptors are invariant under isometry, $\mathbf{C}$ must preserve them, i.e., $\mathbf{CA} = \mathbf{B}$ and 2) the map should commute with the Laplace–Beltrami operator. Our basis functions are eigenfunctions of the Laplace–Beltrami operator, therefore we can express the second requirement as $\Lambda^{\mathcal{N}}\mathbf{C} = \mathbf{C}\Lambda^{\mathcal{M}}$, where $\Lambda^{\mathcal{M}}$ is the diagonal matrix whose diagonal consists of the first $k_{\mathcal{M}}$ eigenvalues of $L_{\mathcal{M}}$, and $\Lambda^{\mathcal{N}}$ is the diagonal matrices whose diagonal consists of the first $k_{\mathcal{N}}$ eigenvalues of $L_{\mathcal{N}}$.

Thus, we obtain the first approximation of $\mathbf{C}$ by solving the optimization problem

$$\mathbf{C} = \arg\min_{\mathbf{X}} \|\mathbf{XA} - \mathbf{B}\| + \varepsilon\|\Lambda^{\mathcal{N}}\mathbf{X} - \mathbf{C}\Lambda^{\mathcal{M}}\|. \tag{6}$$

In our experiments, we took two meshes form the SCAPE dataset. We choose $k_{\mathcal{M}} = k_{\mathcal{N}} = k_d = 80$ and perform L-BFGS to solve Equation (6). The descriptors we chose are HKS function evaluated at different time values.

The topology comes into play in the second phase of the process. While every bijective continuous mapping $f\colon \mathcal{M} \to \mathcal{N}$ gives rise to the corresponding invertible linear map between functional spaces via pullback ($\phi\colon \mathcal{M} \to \mathbb{R}$ maps to $\phi \circ f^{-1}\colon \mathcal{N} \to \mathbb{R}$), the converse is not true. Let us take r connected regions on $\mathcal{M}$ and let Ω_r be the indicator function of the union of the regions. We slightly abuse the notation by writing $\mathbf{C}(\Omega_r)$ for the corresponding function $\mathcal{N} \to \mathbb{R}$ (first, Ω_r needs to be projected onto the corresponding subspace). The authors of [26] show that it is reasonable to require the following: the 0-dimensional persistence diagram of $\mathbf{C}(\Omega_r)$ has exactly as many points as the diagram of Ω_r. In other words, we should simplify the diagram of $\mathbf{C}(\Omega_r)$ to remove all but the first $r-1$ most persistent finite points (we assume $\mathcal{N}$ and $\mathcal{M}$ to be connected, so there is exactly one point at infinity).

We sample $r = 1$ random point and take all points of the mesh that are at most 2 hops away as our region. We want to optimize $\mathbf{C}$ to eliminate all finite points in the diagram of the image of the indicator function $\mathbf{C}(\Omega_1)$. Crucially, unlike the rest of the examples in the paper, the optimization parameters are the entries of matrix $\mathbf{C}$.

The behavior of the diagram loss is shown in Figure 25. The advantage of the critical set method is evident, it rapidly drives the loss to 0.

The vineyards are in Figures 26 to 29.

We should mention that these results are for simplification method that pushes the point (b, d) towards (d, d), i.e., increases the birth values.

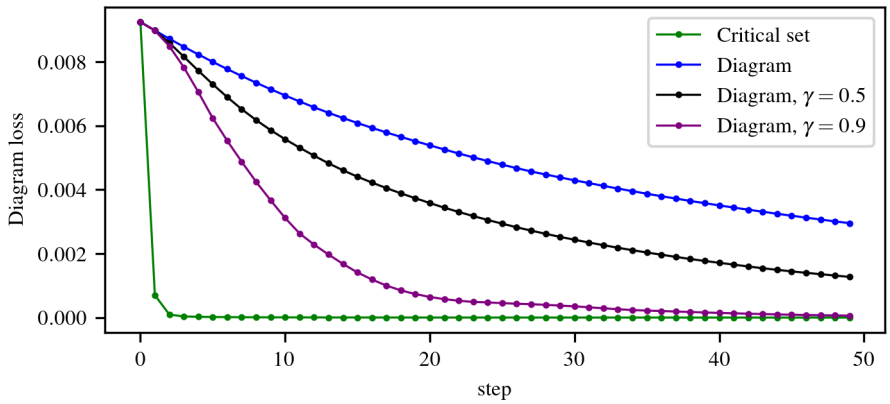

Figure 25: Comparison of the diagram losses during regularization of the functional map $\mathbf{C}$. Diagram methods benefits from momentum, but the critical set significantly outperforms it.

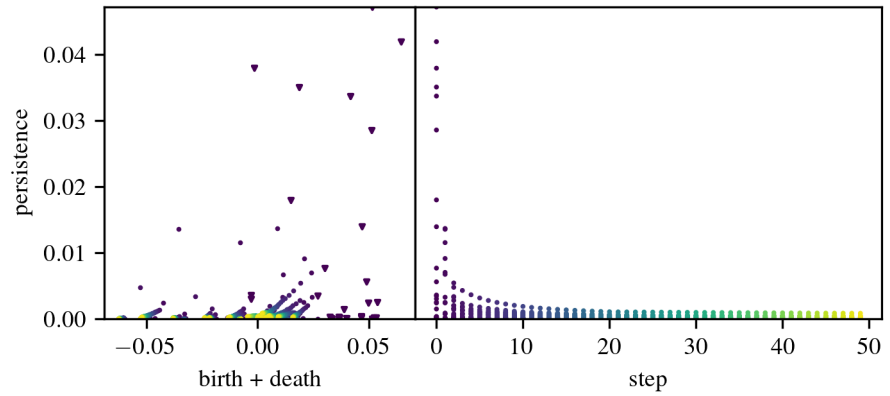

Figure 26: Vineyard of the optimization of the functional map $\mathbf{C}$ guided by the simplification of a superlevel set in a 0-dimensional diagram of $\mathbf{C}(\Omega_1)$, using the **critical set method**. Learning rate is 0.2, without momentum. The color encodes the time step.

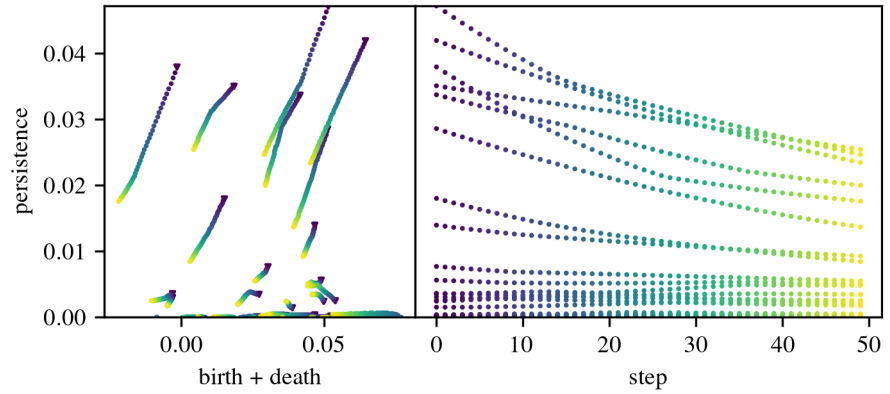

Figure 27: Vineyard of the optimization of the functional map $\mathbf{C}$ guided by the simplification of a superlevel set in a 0-dimensional diagram of $\mathbf{C}(\Omega_1)$, using the **diagram method**. Learning rate is 0.2, no momentum. The color encodes the time step.

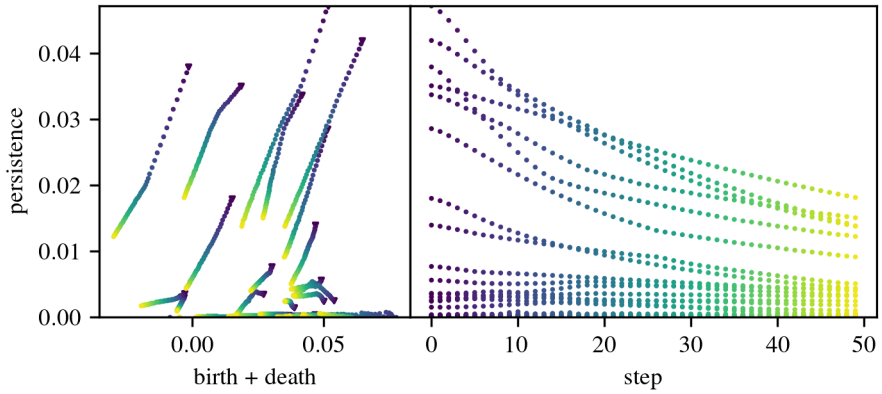

Figure 28: Vineyard of the optimization of the functional map $\mathbf{C}$ guided by the simplification of a superlevel set in a 0-dimensional diagram of $\mathbf{C}(\Omega_1)$, using the **diagram method**. Learning rate is 0.2, momentum $\gamma = 0.5$. The color encodes the time step.

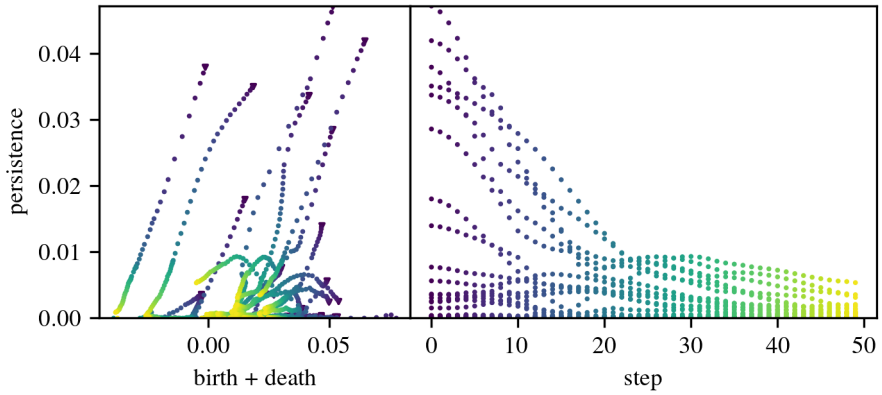

Figure 29: Vineyard of the optimization of the functional map $\mathbf{C}$ guided by the simplification of a superlevel set in a 0-dimensional diagram of $\mathbf{C}(\Omega_1)$, using the **diagram method**. Learning rate is 0.2, momentum $\gamma = 0.9$. The color encodes the time step.

References

[1] Anguelov, D., Srinivasan, P., Koller, D., Thrun, S., Rodgers, J., and Davis, J. Scape: shape completion and animation of people. In *ACM SIGGRAPH 2005 Papers*. 2005, pp. 408–416.

[2] Attali, D., Glisse, M., Hornus, S., Lazarus, F., and Morozov, D. Persistence-sensitive simplication of functions on surfaces in linear time. In *TopoInVis' 09* (2009).

[3] Bauer, U. Ripser: efficient computation of Vietoris–Rips persistence barcodes. *Journal of Applied and Computational Topology* (2021).

[4] Bauer, U., Kerber, M., Reininghaus, J., and Wagner, H. Phat–persistent homology algorithms toolbox. *Journal of symbolic computation 78* (2017), 76–90.

[5] Bauer, U., Lange, C., and Wardetzky, M. Optimal topological simplification of discrete functions on surfaces. *Discrete & computational geometry 47*, 2 (2012), 347–377.

[6] Bendich, P., Edelsbrunner, H., Morozov, D., and Patel, A. Homology and robustness of level and interlevel sets. *Homology, Homotopy and Applications 15*, 1 (2013), 51–72.

[7] Brüel-Gabrielsson, R., Nelson, B. J., Dwaraknath, A., Skraba, P., Guibas, L. J., and Carlsson, G. A topology layer for machine learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)* (2020), pp. 1553–1563.

[8] Carriere, M., Chazal, F., Glisse, M., Ike, Y., Kannan, H., and Umeda, Y. Optimizing persistent homology based functions. In *Proceedings of the 38th International Conference on Machine Learning* (2021), M. Meila and T. Zhang, Eds., vol. 139 of *Proceedings of Machine Learning Research*, PMLR, pp. 1294–1303.

[9] Chen, C., and Kerber, M. Persistent homology computation with a twist. In *Proceedings 27th European Workshop on Computational Geometry* (2011), vol. 11, pp. 197–200.

[10] Chen, C., Ni, X., Bai, Q., and Wang, Y. A topological regularizer for classifiers via persistent homology. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)* (2019), pp. 2573–2582.

[11] Cohen-Steiner, D., Edelsbrunner, H., and Morozov, D. Vines and vineyards by updating persistence in linear time. In *Proceedings of the Annual Symposium on Computational Geometry* (2006), pp. 119–126.

[12] Davis, D., Drusvyatskiy, D., Kakade, S., and Lee, J. D. Stochastic subgradient method converges on tame functions. *Foundations of computational mathematics 20*, 1 (Feb. 2020), 119–154.

[13] de Silva, V., Morozov, D., and Vejdemo-Johansson, M. Dualities in persistent (co)homology. *Inverse problems 27*, 12 (Nov. 2011), 124003.

[14] Edelsbrunner, H., and Harer, J. *Computational topology: an introduction*. American Mathematical Society, 2010.

[15] Edelsbrunner, H., and Morozov, D. Persistent homology. In *Handbook of Discrete and Computational Geometry*. Chapman and Hall/CRC, 2017, pp. 637–661.

[16] Edelsbrunner, H., Morozov, D., and Pascucci, V. Persistence-sensitive simplification functions on 2-manifolds. In *Proceedings of the Annual Symposium on Computational Geometry* (2006), ACM, pp. 127–134.

[17] Edelsbrunner, H., Morozov, D., and Patel, A. Quantifying transversality by measuring the robustness of intersections. *Foundations of Computational Mathematics 11*, 3 (June 2011), 345–361.

[18] Gameiro, M., Hiraoka, Y., and Obayashi, I. Continuation of point clouds via persistence diagrams. *Physica D. Nonlinear phenomena 334* (Nov. 2016), 118–132.

[19] Guo, F., Li, H., Daughton, W., and Liu, Y.-H. Formation of hard power laws in the energetic particle spectra resulting from relativistic magnetic reconnection. *Phys. Rev. Lett. 113* (Oct. 2014), 155005.

[20] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete & computational geometry 28*, 4 (Nov. 2002), 511–533.

[21] Klacansky, P. Open scientific visualization datasets. klacansky.com/open-scivis-datasets/.

[22] Leygonie, J., Carrière, M., Lacombe, T., and Oudot, S. A gradient sampling algorithm for stratified maps with applications to topological data analysis. *arXiv:2109.00530* (2021).

[23] Luo, Y., and Nelson, B. J. Accelerating iterated persistent homology computations with warm starts. *arXiv:2108.05022* (2021).

[24] Morozov, D. *Homological illusions of persistence and stability.* PhD thesis, Duke University, 2008.

[25] Nigmetov, A., Krishnapriyan, A. S., Sanderson, N., and Morozov, D. Topological regularization via Persistence-Sensitive optimization. *arXiv:2011.05290* (Nov. 2020).

[26] Poulenard, A., Skraba, P., and Ovsjanikov, M. Topological function optimization for continuous shape matching. *Computer graphics forum: journal of the European Association for Computer Graphics 37*, 5 (Aug. 2018), 13–25.

[27] Rosenberg, D., Pouquet, A., Marino, R., and Mininni, P. D. Evidence for Bolgiano-Obukhov scaling in rotating stratified turbulence using high-resolution direct numerical simulations. *Physics of fluids 27*, 5 (May 2015), 055105.

[28] Solomon, Y., Wagner, A., and Bendich, P. A fast and robust method for global topological functional optimization. In *International Conference on Artificial Intelligence and Statistics* (2021), PMLR, pp. 109–117.

[29] Tierny, J., and Pascucci, V. Generalized topological simplification of scalar fields on surfaces. *IEEE transactions on visualization and computer graphics 18*, 12 (Dec. 2012), 2005–2013.

[30] Trailie, C. Pyhks, 2018. github.com/ctralie/pyhks.