TOPOSYS
FP7-ICT-2011.8
Contract no.: 318493

www.toposys.org

TOPOSYS

Deliverable D4.2

Progress and Activity Report on WP4

Deliverable Nature:	Report (R)
Dissemination Level: (Confidentiality)	Public (PU)
Contractual Delivery Date:	M24
Actual Delivery Date:	M24
Version:	1.0

Contents

1 Summary

The theory we develop must be relevant for real-world data and systems. In this work package we develop applications which verify our approaches. In the second year, we continued with the robotics applications as well as considered other types of data. In particular, we considered the following problems:

1. Multiscale Topological Trajectory Classification
2. Application of Persistence to Complex Data

We have begun experiments on social media data including Wikipedia in different languages, metadata from the ArXiV preprint server, and Twitter data (for example, we can construct dynamic co-occurence graphs). Unfortunately, we have found two important factors:

- The current algorithm for the self-map does not scale well enough.
- The simple metric Rips constructions are too coarse.

To remedy the situation, we have begun developing a better algorithm for the self-map, where we anticipate that the implementation will be ready before the end of the calender year (but it is not ready in time for this report). For the second, we have begun using Cech complexes as well as other information based metrics. Furthermore, the Delauanay-Cech work described in WP1 will also aid in both the accuracy and scalability of the methods.

In the meantime, we have conducted experiments on other types of complex data including genetic data and data obtained from Influenzanet, which tracks outbreaks of the flu throughout Europe. These experiments were reported in workshops - and work will continue in the coming year.

Finally we also present work in making computation of persistent homology distributed. This includes engineering work that illustrates that distributed computation can be efficiently done - where additional work will be done in the final year.

The contents of this report are based on two papers which have been published and two prelimary reports which have been presented at two workshops).

1. Multiscale Topological Trajectory Classification - RSS 2014
2. Distributed Homology Computation - ALENEX 2014
3. A Topological Data Analysis Approach to Epidemiology - ECCS 2014
4. Topological Structures In Gene Expression Data - Genetic Analysis Workshops 2014

2 Multiscale Topological Trajectory Classification with Persistent Homology

The problem of determining a continuous path $\gamma : [0,1] \to \mathcal{C}_f$ between two points in the collision free subset $\mathcal{C}_f$ of some configuration space $\mathcal{C} \subseteq \mathbb{R}^d$ is an important classical motion planning problem. Since, in realistic robotics applications, an explicit description of $\mathcal{C}_f$ is often not available, popular algorithms such as Rapidly-exploring Random Trees (RRT) and Probabilistic Roadmaps (PRM) [25, 24, 20] are based on the idea of utilizing a set of random samples $X \subset \mathcal{C}_f$ to construct a graph Γ with vertices in $\mathcal{C}_f$ and where edges correspond to local paths which can be determined by a local path planner. The graph Γ can then be used to efficiently carry out motion planning. If $\mathcal{C}_f$ is a tame space, Γ, for sufficiently large X, provides an approximation of $\mathcal{C}_f$ which allows us to answer basic questions about the path-connectivity of $\mathcal{C}_f$. However, Γ *does typically not capture higher order homological information.*

We give a novel approach based on filtrations $\mathcal{F} = \{\mathcal{F}_r : r \geq 0\}$ of simplicial complexes defined in terms of random samples $X \subset \mathcal{C}_f$. From such filtrations, we then extract higher-order topological information for

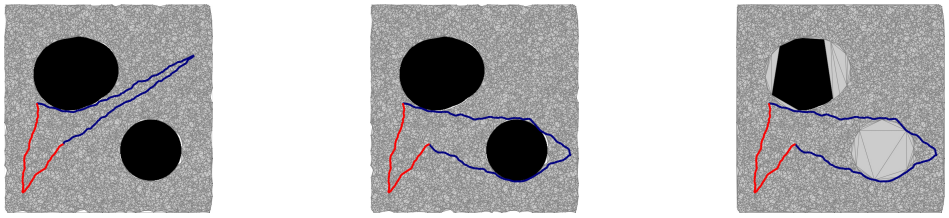


Figure 1: We display a rectangular configuration space $\mathcal{C}$ of side-length 500 and with two obstacles (in black). Several trajectories are depicted in red and blue. The gray area displays an approximation of $\mathcal{C}_f$ by a Delaunay-Čech complex $DC_r(X)$ from 10000 samples $X \subset \mathcal{C}_f$ and with filtration value $r = 11.07$ (left and middle figure) and $r = 73.76$ (right figure). All of our figures are best viewed in color.

the purpose of understanding and classifying equivalence classes of trajectories in $\mathcal{C}_f$. Given a sufficiently good approximation of $\mathcal{C}_f$ by $\mathcal{F}_r$, our approach yields a finite set of equivalence classes with the property that *no trajectory belonging to one equivalence class can be continuously deformed to any trajectory in any of the other equivalence classes*. Our filtration $\mathcal{F}$ is based on Delaunay-Čech complexes which depend on a scale parameter r and which are descirbed in the report for WP1 to provide a homotopy-equivalent reconstruction of the space $X_r = \bigcup_{x \in X} \mathbb{B}_r(x)$ [3], where $\mathbb{B}_r(x) = \{y \in \mathbb{R}^d : \|x - y\| \leq r\}$. Our work utilizes persistent homology [15, 11, 16] which generalizes classical homology groups to a multiscale setting – meaning that we are able to compute topological information about the analogue $\mathcal{F}_r$ of Γ for all scales $r \geq 0$ simultaneously without having to choose a particular scale upfront. Additionally, the 1-skeleton $\mathcal{F}_r^1 \subseteq \mathcal{F}_r$ is a graph which can be used for path-planning. When a cost function $c : \mathcal{C}_f \to \mathbb{R}$ is defined, we furthermore study the space of paths in $M_{r,\lambda} = X_r \cap c^{-1}(-\infty, \lambda]$ and show how resulting path classes can be obtained. We evaluate our approach in 2, 3, 4, and 6 dimensions in simulation and using a Baxter robot.

2.1 Motivation and Related Work

For a robot to reason efficiently about trajectories within its own free configuration space $\mathcal{C}_f$, or about the motions of other human or robot agents in its environment, a suitable partitioning of continuously varying families of trajectories into a discrete set of equivalence classes is desirable.

Clustering trajectories is difficult in general since trajectories can have varying length and are not immediately representable as vectors in a vector space of fixed dimension as required by commonly used algorithms. Several approaches to the classification of trajectories, as reviewed in [33], are based on various approaches to measuring the dissimilarity between trajectories, such as the Hausdorff distance, edit distances and dynamic time warping. For the purpose of activity analysis, the work of [29] reviews trajectory clustering approaches based on various clustering algorithms and distance measures.

In robotics, the knowledge of classes of trajectories is beneficial for example in the learning by demonstration framework [8] where movement primitives of a robot's behavior are constructed from initial trajectory demonstrations provided by a human teacher. Equivalence classes of robot trajectories can furthermore be useful in order to reason about alternative trajectories when a subset of trajectories becomes invalid due to changing environment conditions. The recent work [22] has demonstrated the usefulness of trajectory classes of local paths to improve the efficiency of a motion planning algorithm in particular. Purely topological approaches to the analysis of trajectories in $\mathcal{C}_f$ focus on notions of equivalence which do not depend on a metric. There, two paths $\alpha, \beta : [0, 1] \to \mathcal{C}_f$ with $\alpha(0) = \beta(0)$, $\alpha(1) = \beta(1)$ are called homotopy equivalent if α can continuously be deformed to β in $\mathcal{C}_f$ while keeping the end-points fixed. We use Fig. 1 as a running example for illustration. Note that all figures in this paper are best viewed in color. For now, consider $\mathcal{C}_f$ as being approximated by the gray region, while the black regions correspond to obstacles. Several trajectories with identical start and end-points are depicted in red and blue. The blue trajectory in the leftmost figure is homotopy equivalent to the red trajectory, while the trajectories in the middle figure are not homotopy equivalent. Note that, while trajectories can have identical distance in $\mathbb{R}^2$ under *e.g.* the Hausdorff distance, they may or may not be homotopy equivalent. When the lower obstacle is removed in the right figure, the red and blue paths from the middle figure become homotopy equivalent, for example.

Topological concepts such as retractions and cell decompositions have played a key role in classical

approaches to motion planning [23]. There, $\mathcal{C}_f$ is typically assumed to have a known algebraic or semi-algebraic structure. The visibility graph in 2D and retraction-based methods rely on constructing a graph using which motion planning is performed. The general roadmap method of [10] uses ideas closely related to Morse theory and projections to lower-dimensional spaces to obtain a complete motion planner for semi-algebraic sets. Similarly, the seminal work of [31] proceeds by constructing an exact cell decomposition by means of a cylindrical algebraic decomposition of $\mathcal{C}_f$. This is related to our approach since our simplicial complexes form a particular type of approximate cell decomposition. In [31], homology groups of $\mathcal{C}_f$ are computed by an exact cell decomposition $\mathcal{Z}$ and the general path planning problem is solved using $\mathcal{Z}$. These classical works have to the best of our knowledge however not considered the use of the first homology group of $\mathcal{C}_f$ for trajectory classification, and the focus has been on motion planning and not classification. An important difference to our work is the fact that we only assume the knowledge of *potentially noisy point-samples* from $\mathcal{C}_f$ using which we build a *simplicial complex filtration* rather than assuming a known description of $\mathcal{C}_f$ as a (semi-)algebraic set. Furthermore, our approach allows us to study the homotopy equivalence of paths within the neighborhood X_r of a set of samples $X \subset \mathcal{C}_f$ more generally, *e.g.* when X_r does not yield a reconstruction of the full space $\mathcal{C}_f$. In a more recent related work, [32] construct an approximate cell decomposition using a recursively refined decomposition of $\mathcal{C}_f$ into hypercubes to ensure a sufficiently fine reconstruction of $\mathcal{C}_f$. However, only the path-connectivity of this decomposition is then used for motion planning and homological properties are not further investigated.

Our work is also related to sampling-based algorithms constructing a graph Γ from X to answer questions about the path-connectivity of $\mathcal{C}_f$. RRTs and PRMs [25, 24, 20], in particular, are examples of these which have attracted unabated interest since their invention [26, 27, 19]. The graph Γ can be thought of as an approximation of $\mathcal{C}_f$ from X. The filtrations of simplicial complexes used in our work extend the concept of a graph to a multiscale approach which can recover more detailed information about $\mathcal{C}_f$. Such filtrations depending on a scale parameter r have been used in topological data analysis (TDA) [15, 11, 16] to study the persistent homology groups which we use here and which capture information about the topology of data at all scales simultaneously.

One of the advantages of the knowledge of homotopy classes is that a motion planning algorithm can utilize efficient replanning within each such class [9]. Since local variational or gradient based methods can continuously deform trajectories towards local optimality, it is advantageous to maintain a set of homotopy inequivalent trajectories each of which can then be optimized using these methods. Topological information about path classes hence allows us to incorporate non-trivial *global* information with these *local* methods.

Recent approaches which attempt to obtain equivalence classes of paths include the works [18, 17] on path deformation roadmaps where a graph-based representation to plan in the space of paths up to a class of continuous deformations is proposed. Recently, researchers have in particular investigated homotopy classes of trajectories in explicitly described spaces. Using the classical residue theorem of complex analysis [5] studied an application of *homology classes* to motion planning in 2D in the case where the obstacles in $\mathcal{C}$ can be contracted into representative and explicitly defined points. In [6], this was extended using electromagnetism theory and Ampère's law to the case of 3D. There, obstacles were assumed to be contracted into skeletons and then modeled as current-carrying wires. Similarly to our work, the authors argue that homological information is useful and computationally favorable to more general homotopy invariants in robotics. In [7], a generalization to arbitrary dimension is proposed and an integration of differential 1-forms over cycles is shown to be sufficient to determine topological classes using the classical language of de Rham cohomology theory. In [21], motion planning in 2D with homology constraints is formulated as a mixed-integer quadratic program by endowing path segments with binary labels that identify their relation to the domain obstacles. A problem the above recent approaches suffer from is that *they require an explicit description of the obstacles in the configuration space*, e.g. in 2D as unions shapes each of which is contractible to a geometrically specified point $p \in \mathcal{C} - \mathcal{C}_f$. In many cases, such information is however not easily available for real robotic systems or too expensive to compute. We instead propose a *data-driven, sampling-based approach* to building a representation of $\mathcal{C}_f$ from which topological information about trajectories can be extracted.

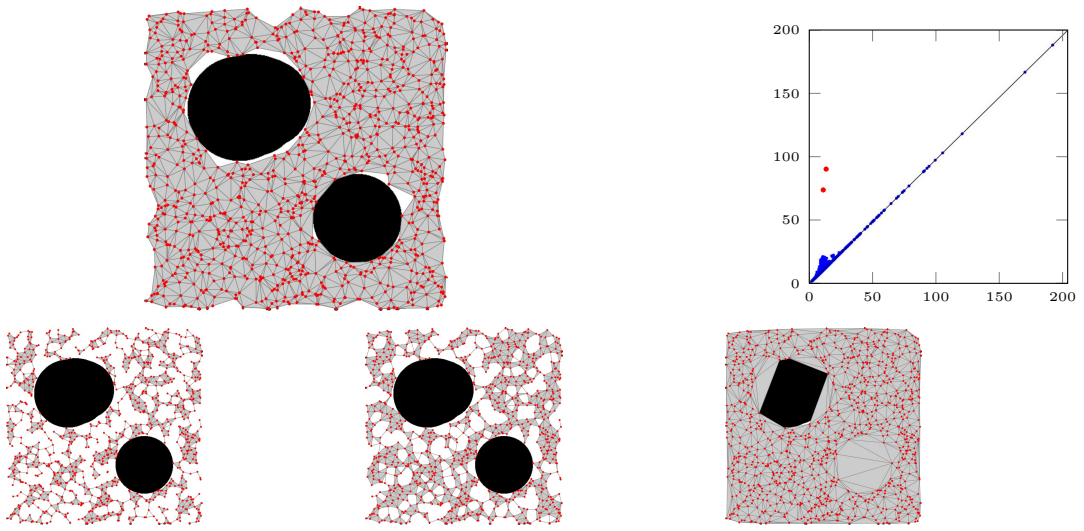


Figure 2: A reconstruction of $\mathcal{C}_f$ from 1000 samples (red points) on a square of side-length 500. $DC_{25}(X)$ is displayed in the top-left yielding a good approximation to $\mathcal{C}_f$. The first persistence diagram for $DC_r(X)$ is shown in the top-right. The two marked red points $p_1 = (10.58, 74.0)$, $p_2 = (12.97, 90.38)$ with large persistence correspond to the birth and death filtration of the two holes in $\mathcal{C}_f$. The bottom row displays $DC_{10.58}(X)$, $DC_{12.97}(X)$ and $DC_{74.0}(X)$ which correspond to the birth of the smaller and larger hole (the first time they are enclosed by edges), and finally to the death filtration value of the smaller hole (the hole is covered at $r = 74.0$).

2.2 Theoretical Background

Filtrations and persistent homology provide a key tool to determine multiscale topological properties. We do not review the concepts, which have been reviewed numerous times in this project but refer to [15, 16]. For Delaunay-Čech Complexes, we refer to [3] and WP1 where we report on this work explicitly.

In Fig. 2, we display the diagram for $p = 1$ and $DC_r(X)$. Observe that the two obstacles correspond to the two red points in the diagram which are far from the diagonal. The remaining points correspond to holes which are due to noise and which do not persist for a large filtration interval. Note also that, for $p = 0$, the persistence diagram measures the merging of connected components of $DC_r(X)$ as r is increased [15].

$H_1(Y)$ and homotopy classes of trajectories The only piece of background work we explicitly cover is the connection between the first homology group and homotopy classes of paths in a topological space Y. The obvious case to keep in mind is $Y = \mathcal{C}_f \subset \mathbb{R}^d$. Recall that the first fundamental group $\pi_1(Y, x_0)$ [16] is a well-known group whose elements consist of equivalence classes of closed continuous curves through $x_0 \in Y$ and lying entirely in Y. Two closed paths $\alpha, \beta : [0, 1] \to Y$ through x_0 lie in the same equivalence class if there exists a homotopy (i.e. a continuous deformation) between them which is constant at the base-point x_0. When Y is path-connected, $\pi_1(Y, x_0)$ is independent of the chosen base-point x_0 and hence often denoted simply by $\pi_1(Y)$. Furthermore, if the spaces Y, Y' are homotopy equivalent spaces, $\pi_1(Y)$ and $\pi_1(Y')$ are isomorphic as groups. Two paths γ_1, γ_2 in Y with the same start point x and end point y can be deformed into each other via a homotopy if the closed curve γ following γ_1 from x to y and then γ_2 from y to x is trivial in $\pi_1(Y)$. Hence, $\pi_1(Y)$ is a natural group to consider for the purpose of trajectory classification. Unfortunately, to the best of our knowledge, no sufficiently efficient method for general configuration spaces exists to compute the group structure of $\pi_1(Y)$ which can be complicated and non-commutative. To extract topological information about homotopy classes, we can turn to the first singular homology group $H_1(Y)$ with binary $\mathbb{Z}_2 = \{0, 1\}$ coefficients, yielding a vector space which can be explicitly computed via simplicial homology when Y is homotopy equivalent to a simplicial complex $\mathcal{K}$. The closed curve γ can be represented explicitly as a 1-cycle in a sufficiently fine subdivision of $\mathcal{K}$ when a deformation retraction from Y to $\mathcal{K}$ is computable, and γ then corresponds to a vector $[\gamma]$ in $H_1(Y) \cong H_1(\mathcal{K})$. Finally, $[\gamma] \neq 0$ implies that γ_1 and γ_2 are not homotopy equivalent, allowing us to discern homotopy classes of continuous paths. Note however that homology is a weaker concept than homotopy, so $[\gamma] = 0 \in H_1(Y)$ does not imply that γ_1

and γ_2 are homotopy equivalent. To gain somewhat more granularity, one can further replace $\mathbb{Z}_2$ coefficients for example with $\mathbb{Z}_p$ coefficients for a large prime p. In this work we choose $\mathbb{Z}_2$ coefficients due to their computational advantages for large simplicial complexes.

2.3 Methodology

We consider a configuration space $\mathcal{C} \subset \mathbb{R}^d$ and the set $\mathcal{C}_f \subseteq \mathcal{C}$ of collision-free configurations. We do not assume that we have an explicit description of $\mathcal{C}_f$ or $\mathcal{C}$ available, and we would like to study homotopy classes of a set of trajectories $T = \{\gamma_1, \ldots, \gamma_k\} \subset \mathcal{C}_f$ with a fixed starting point $x \in \mathcal{C}_f$ and end point $y \in \mathcal{C}_f$. In order to classify the trajectories, we shall exploit the connection between homotopy classes and the first homology group which we just discussed. We now consider two multiscale settings:

1) X is a sufficiently dense sample We assume that $X = \{x_1, \ldots, x_n\} \subset \mathcal{C}_f$ yields a sufficiently dense sample, for example sampled via rejection sampling from the uniform distribution on $\mathcal{C}$, or via a randomized exploration of the configuration space. We can then ask about a likely approximation of $\mathcal{C}_f$ from X. Our working hypothesis is that the family of spaces $\{X_r = \bigcup_{x \in X} \mathbb{B}_r(x) : r > 0\}$ contain good such estimates. If X was sampled uniformly and $\mathcal{C}_f$ is a smooth compact submanifold $M \subset \mathbb{R}^d$, this intuition is in fact well-founded due to the reconstruction theorem of [30] which guarantees that, for a sufficiently dense sample set, X_r deformation retracts to the manifold M for appropriately chosen r. Using the previously introduced Delaunay-Čech complex and the fact that $DC_r(X)$ is homotopy equivalent to X_r [3], we will then compute homological information about X_r from $DC_r(X)$.

2) $X \subset T$ We assume only the availability of the trajectories T. We then discretize each trajectory γ_i as a piecewise linear curve and use the vertex positions of all the piecewise linear segments in T as our sample set X. We study the homotopy classes of these trajectories *within the topological spaces* X_r which constitute an approximation of the r-neighborhoods around T. This then allows us to classify trajectories within X_r. In this framework, holes can arise either due to obstacles in the configuration space (as in the dense case), or due to the distribution of the trajectories in $\mathcal{C}_f$. We consider applications of this case in our experiments with a Baxter robot.

For a sample set X, let R be the minimal $r > 0$ such that $\gamma_i \subset X_r$ for all $i \in \{1, \ldots, k\}$. Our approach in both cases above will now be to study the homotopy classes of these paths *in the topological spaces* $X_r \simeq DC_r(X)$, *for* $r \geq R$.

Trajectory Discretization In order to compute properties of a trajectory $\gamma : [0, 1] \to \mathcal{C}_f$, we first need to represent γ by a homotopy equivalent path of edges (*i.e.* 1-simplices) in $DC_R(X)$. A fast heuristic procedure for this is to consider $v_i = \gamma(i/N)$, for some large $N \in \mathbb{N}$, to map v_i to a closest 0-simplex $v_i' \in DC_R(X)$ and to then replace the path segment between v_i, v_{i+1} by a shortest edge-path between v_i' and v_{i+1}'. Alternatively, one can attempt to construct an explicit deformation retraction from X_R to $DC_R(X)$ mapping γ first to a path contained in $DC_R(X)$ and then approximating γ by a homotopy equivalent sequence of 1-simplices on a sufficiently fine subdivision of $DC_R(X)$. The Alpha complexes $A_r(X)$ of [14] are subcomplexes of $DC_r(X)$ for all $r > 0$ which are also homotopy equivalent to X_r and onto which an explicit such deformation retraction from X_r has been described in [14], for example. While the study of efficient and theoretically sound homotopy equivalent trajectory discretizations should be explored further, we will instead focus on the classification problem here, assuming that each trajectory has been discretized as a path of edges in $DC_R(X)$.

Homological Trajectory Classification Consider a set of edge-paths $\{\alpha_0, \ldots, \alpha_m\}$ in $DC_R(X)$ starting and ending at 0-simplices $s, t \in DC_R(X)$ respectively. We consider the 1-cycle $c_{\alpha_0}(\alpha_u) \stackrel{\text{def}}{=} \alpha_0 + \alpha_u \in Z_1(DC_R(X))$. Now $[c_{\alpha_0}(\alpha_u)] \neq [c_{\alpha_0}(\alpha_w)] \in H_1^{i,j}(DC(X))$ implies $[\alpha_u + \alpha_w] \neq 0$, so that α_u, α_w are not homotopy equivalent in $DC_r(X), R \leq r_i \leq r < r_j$, where $r_1 < \ldots < r_m$ denote the critical filtration values at which $DC_r(X)$ changes. We hence have trajectory classes $\{[c_{\alpha_0}(\alpha_0)], \ldots, [c_{\alpha_0}(\alpha_m)]\} \in H_1^{i,j}$ and the class

membership can be computed once we have determined a basis for $H_1^{i,j}$. Note that α_0 corresponds to the zero vector $0 = [c_{\alpha_0}(\alpha_0)]$ and there can be up to 2^k trajectory classes for fixed s, t and i, j when $dim(H_1^{i,j}) = k$. We can now compute a basis for $H_1^{i,j}$:

Lemma 1. *Let $K_0 \subset \ldots \subset K_n$ be a simplexwise filtration of simplicial complexes, let $R = DV$ denote the reduced boundary matrix after applying the left-to-right reduction algorithm, and let $E_p \subseteq E$, $P_p \subseteq P$ denote those elements corresponding to p-cycles only. For $i \leq n$, a basis of $Z_p(K_i)$ is given by $S^i = \{R_t : (s,t) \in P_p, s \leq i\} \cup \{V_s : s \in E_p, s \leq i\}$, and, for $i \leq j \leq n$, the image of the set*

$$T^{i,j} = \{R_t : (s,t) \in P_p, s \leq i, t > j\} \cup \{V_s : s \in E_p, s \leq i\}$$

in $H_p^{i,j} = Z_p(K_i)/(B_p(K_j) \cap Z_p(K_i))$ forms a basis of $H_p^{i,j}$. Finally $\#E_p = dim(H_p(K_n))$.

Proof. This follows from the reduction algorithm [16]. □

In order to classify $\{\alpha_0, \ldots, \alpha_m\}$, we first select a simplex-wise refinement $\{K_i\}_{i=1}^n$ of the filtration given by $DC_r(X)$, $r \geq 0$. Next, we compute the $\mathbb{Z}_2$ coordinates of $c_{\alpha_0}(\alpha_u)$ for $0 \leq u \leq m$ in the basis S^n once. To classify trajectories *at a scale given by the filtration value* $r_i = f(\sigma)$, we simply look up the binary coordinates of $c_{\alpha_0}(\alpha_u)$ restricted to the basis elements $T^{i,i} \subseteq S^n$. Similarly, we can check if two trajectories α_u, α_w are homotopy inequivalent for all $r_i \leq r < r_j$ by looking up whether the coordinates of $c_{\alpha_0}(\alpha_u)$ and $c_{\alpha_0}(\alpha_w)$ differ in the basis $T^{i,j} \subseteq S^n$.

Note now that $DC_r(X) = D(X)$ for sufficiently large r, where $D(X)$ denotes the full Delaunay triangulation, and $H_1(D(X)) = \{0\}$ since $D(X)$ is contractible. Hence E_1 is empty implying that we *do not need to keep track of the matrix V* to determine a basis of $H_1^{i,j}$. This is important since, in our experiments, these matrices have millions of columns and R is typically very sparse and of low rank, while V has full rank. Since *low* is injective on the set S^n, we order elements of S^n (for $p = 1$) by their *low* value and we store $low^{-1} = l$ as a map such that $l(k)$ is that element $s \in S^n$ with $low(s) = k$. For any cycle $c \in Z_1(K_i)$, we can then trivially solve for the coefficients in the basis S^n by iterating $c \leftarrow c + l(low(c))$. Each iteration reduces $low(c)$ until we arrive at the zero vector. In the ordered basis S^n, c then has non-zero coefficients $F(c) \in \mathbb{Z}_2^{\#S^n}$ exactly at those basis elements $s \in S^n$ for which $low(s) = low(c)$ during the execution of the above loop. Again, n can be very large (millions), but the vector $F(c)$ is in our experiments very sparse so that the algorithm does not exhibit its worst cast $O(n^2)$ computation time. We call $F(c_{\alpha_0}(\alpha_u)) \in \mathbb{Z}_2^{\#S^n}$ the *persistent cycle coordinates* of $\alpha_u \in Z_1(K_i)$ with respect to α_0.

If we want to determine a trajectory class at scales corresponding to filtration values $r_i < r_j$, we select the coordinates $F^{i,j}(c_{\alpha_0}(\alpha_u))$ of $F(c_{\alpha_0}(\alpha_u))$ corresponding to the basis $T^{i,j}$. Two trajectories α_u, α_w are then not homotopy equivalent if $F^{i,j}(c_{\alpha_0}(\alpha_u)) \neq F^{i,j}(c_{\alpha_0}(\alpha_w)$. Each non-zero coordinate of $F(c_{\alpha_0}(\alpha_u))$ corresponds to a column R_t of R which has a death filtration value $f(\sigma_t)$. At filtration value r, only those non-zero coordinates that have not died yet contribute to the classification of cycles. We hence obtain an agglomerative clustering of trajectories lying in a common $DC_R(X)$ as we increase the filtration value $r \geq R$. Finally, at r_m, $DC_{r_m}(X) = DC_\infty(X) = \mathrm{Conv}(X)$ and all trajectories then lie in the same class.

Illustration Consider Fig. 1. The red trajectory corresponds to α_0 and the two blue trajectories in the left and middle figure represent α_1, α_2 respectively, and all trajectories lie in $DC_{r_i}(X)$, $r_i = 11.07$. We have $[c_{\alpha_0}(\alpha_0)] = [c_{\alpha_0}(\alpha_1)] = 0 \in H_1^{a,b}$ for all $i \leq a \leq b$, but $[c_{\alpha_0}(\alpha_2)] \neq 0 \in H_1^{a,b}$, for $i \leq a \leq b \leq j$, where $r_j = 73.76$ is the critical filtration value at which the hole surrounded by α_0, α_2 gets filled in.

2.4 Filtrations with Cost Functions

Suppose now that we have sampled $\mathcal{C}_f$ sufficiently densely and that $DC_R(X)$, for some fixed R, provides a good approximation of $\mathcal{C}_f$. Consider a cost-function $c : \mathcal{C}_f \to \mathbb{R}$. Our aim now is not only to classify trajectories in the space $\mathcal{C}_f$, but to take into account a threshold for the cost function. Given c, we define the cost $\hat{c}$ of a k-simplex $\sigma = \{v_0, \ldots, v_k\} \in DC_R(X)$ to be $\hat{c}(\sigma) = \max(c(v_0), \ldots, c(v_k))$. Then $\hat{c}$ satisfies $\hat{c}(\tau) \leq$

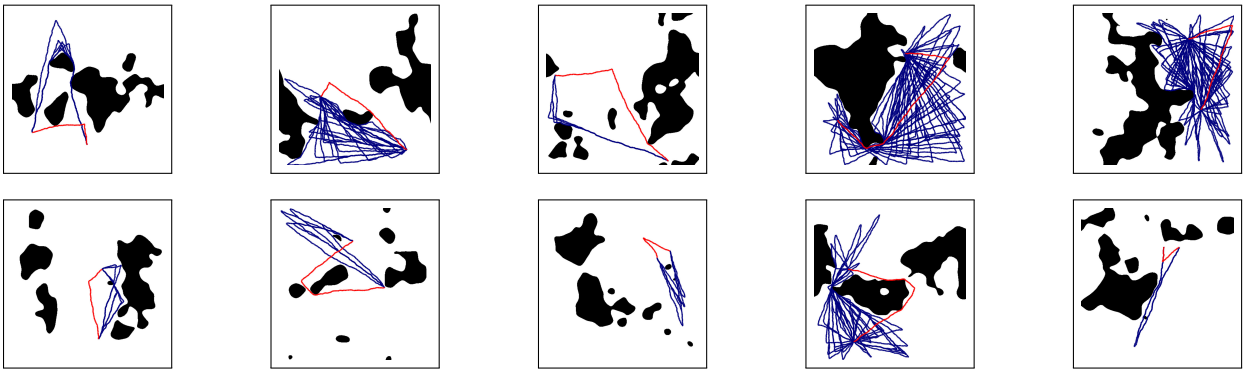


Figure 3: We display example worlds and examples of paths which were determined to lie in a single class (in blue) at filtration value r_2. $DC_{r_2}(X)$ was constructed from 100000 samples and the classes are computed using the indicated red trajectories corresponding to α_0.

$\hat{c}(\sigma)$ whenever $\tau \subseteq \sigma$. In order to apply our algorithm without having to keep track of the potentially non-sparse matrix V, we furthermore let $\hat{c}(\sigma) = \max(c(v_0), \ldots, c(v_k)) + C$ for any k-simplex $\sigma \in D(X) - DC_R(X)$ and for C larger than the cost of any $v \in X$. Then $L_{R,\lambda} = \hat{c}^{-1}((-\infty, \lambda])$ yields a filtration as λ varies and $L_{R,\lambda} \subseteq DC_R(X)$ for $\lambda < C$ and $L_{R,\infty} = D(X)$, ensuring $H_1(L_{R,\infty}) = 0$. We think of $L_{R,\lambda}$ as an approximation to $M_{R,\lambda} = X_R \cap c^{-1}((-\infty, \lambda])$ for $\lambda < C$.

2.5 Experiments

Our experiments were performed on an Intel Core i7 laptop with 8GB of RAM. We present only the computation times of core algorithms and disregard the time required to load data into memory. We used the matrix and binary tree column vector data structure of the PHAT library [4] to efficiently manipulate large boundary matrices. Instead of working with the full simplicial complex $DC_r(X)$, we extracted the 2-skeleton $DC_r^2(X)$ from the Delaunay triangulation $D(X)$. The 2-skeleton is sufficient for our purposes since $H_1^{i,j}(DC_r(X))$ does not depend on higher dimensional simplices. We reduced only the submatrix of the boundary matrix corresponding to the first homology group. $D(X)$ was computed with CGAL [1] for all but our Baxter experiments where we used QHull [2] which was faster in higher dimensions, but which lacks the numerical robustness of CGAL. A supplementary video can be found at `http://www.csc.kth.se/fpokorny`.

Trajectory classification in 2D We generated the set of 2D worlds $W_1, \ldots, W_{10}$ displayed in Fig 3 and of size 512×512 by sampling Gaussian Random Fields and defining those regions above a threshold to be obstacles. From the resulting free space $\mathcal{C}_f$, we sampled $N \in \{1000, 10000, 100000, 1000000\}$ uniform samples. We computed the Delaunay-Čech filtration for all examples and recorded the computation times of the Delaunay triangulation, for the construction of the filtration, as well as the time required to reduce the boundary matrix D to its reduced form R. The Delaunay triangulation took 1ms, 2ms, 76ms, 810ms, the construction of the filtration took 11ms, 31ms, 278ms and 3.27s and the reduction of the boundary matrix took 14ms, 13ms, 76ms, 981ms on average as the sample size increased. We investigated the filtration $DC_r(X)$ at various thresholds. At a filtration value of $r_1 = 25\sqrt{1000/N}$, we found that $\mathcal{C}_f$ was conservatively covered, while at $r_2 = 35\sqrt{1000/N}$, the space was well covered with a minimum number of holes in collision free areas. In order to investigate interesting path classes, we generated a set of 1000 paths per world and sample setting as follows: In 10 trials, we selected two sample points v_1, v_2 at random and, for each such setting, we selected another 100 random waypoints $w_1, \ldots, w_{100}$ from the sampled point-cloud. We determined shortest edge-path from v_1 to w_i and then to v_2 utilizing Dijkstra's algorithm on the 1-skeleton graph of $DC_{r_1}(X)$. The computation times for the *persistent cycle coordinates* for these paths were 1.8ms, 10ms, 115ms and 1.75s for a batch of 100 query paths and for the respective sample sizes on average. These encouraging timings suggest that our framework could be used as a classification 'black box' *e.g.* for continuous trajectory optimization

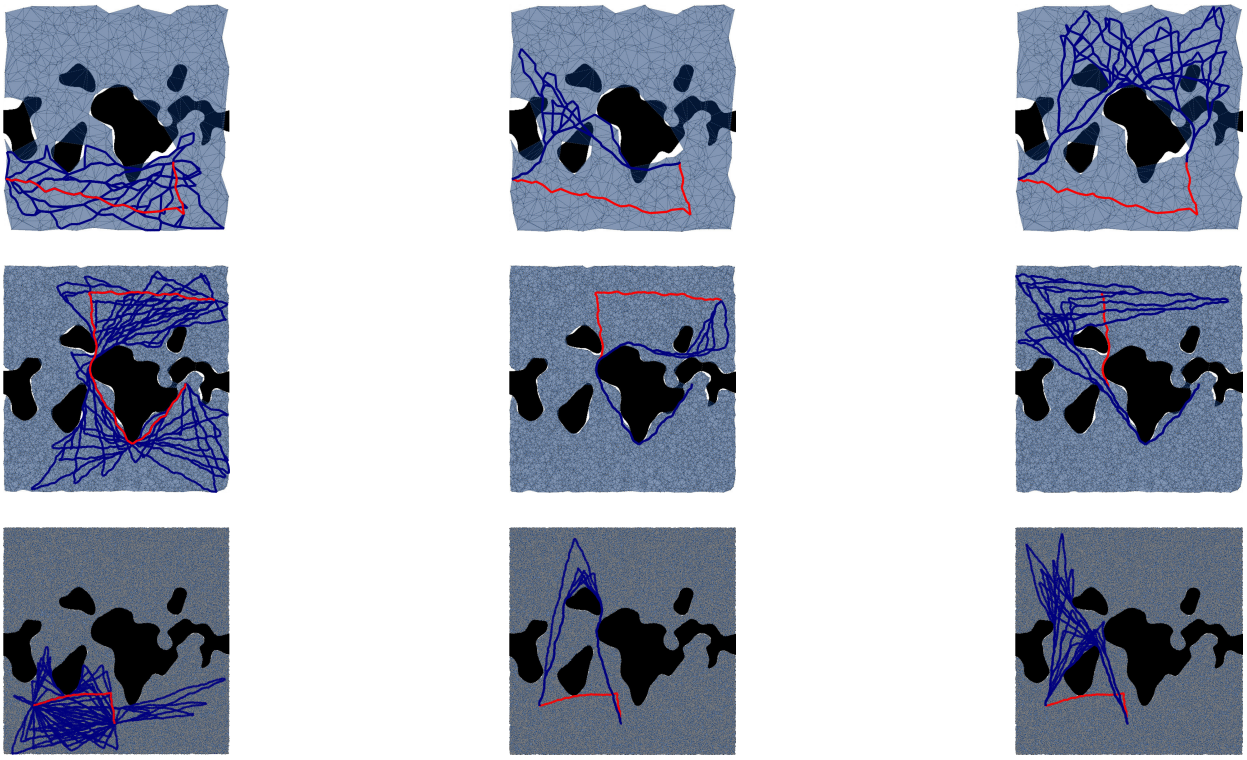


Figure 4: We display the example world W_1 with $DC_{r_2}(X)$ for 1000, 10000 and 100000 sample points per row. In each column, we plot paths $\alpha_1, \ldots, \alpha_s$ (in blue) which belong to a fixed trajectory class at filtration value r_2. The fixed reference path α_0 is plotted in red. As expected, we can clearly see that two paths in different classes also lie in different homotopy classes. In our experiments, paths within a class are furthermore homotopy equivalent in $DC_{r_2}(X)$, but the quality of the approximation $DC_{r_2}(X) \simeq \mathcal{C}_f$ is only sufficient for 10000 or more sample points as can be seen in the right figure in the first row. There, some 2-simplices (triangles) cover the thin obstacle region to the right.

engines.

Trajectory classification in 4D We consider the planar robot arm displayed in the top left of Fig. 5 attached to the central black disk and with 4 joints $\theta_1, \ldots, \theta_4$. We constrain $\theta_1 \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, $\theta_2, \theta_3, \theta_4 \in [-0.9\pi, 0.9\pi]$ and furthermore disallow self-collisions and collisions with the environment (the black rectangle and the floor), yielding $\mathcal{C}_f \subset \mathbb{R}^4$. The robot now has the task of moving from the start configuration displayed in blue to the red goal joint configuration as shown in the top left figure. We sampled 100000 poses uniformly in $\mathcal{C}_f$ using OpenRave [13] and applied our framework. $DC^2_\infty(X)$ had about 6.2 million triangles and 1.8 million edges. The right part of Fig. 5 displays the resulting first persistence diagram which clearly shows that a single homological feature has large persistence in $\mathcal{C}_f$. The projection of the joint configurations onto the first two angles, as shown in the middle figure, confirms the existence of a single hole. We computed 1000 edge-paths in $DC_{0.25}$ between the start and end-configuration using 1000 random waypoints as before. For filtration values $r \in [0.301, 0.382]$ only two trajectory classes existed. The reduction of the boundary matrix took 0.46s, while the persistent cycle coordinates for all 1000 paths were calculated in 0.55s. The Delaunay triangulation in $\mathbb{R}^4$ took 251s, partially due to the increased dimension. Note however that these results are not directly comparable to the 2D case since methods for 2D Delaunay triangulations in CGAL [1] are especially optimized. We inspected the trajectories in each homology class and found that they were classified according to whether the second link was positioned to the left or to the right of the base link of the arm when $\theta_1 = 0$ as the arm passed the narrow passage (see the bottom left part of Fig. 5). Our framework hence allows the robot to *discover* the fact that two fundamentally different solution trajectory

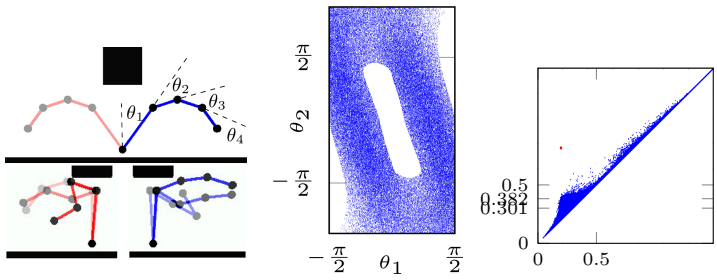


Figure 5: The top left figure shows the robot arm in start configuration (blue) on the right and in goal configuration (red) on the left. The right figure displays the first persistence diagram for our reconstruction with one red point far above the diagonal. A projection of the samples onto θ_1, θ_2 is shown in the middle and an illustration of the difference between the two trajectory classes for $r \in [0.301, 0.382]$ is shown in the bottom left figure. In the first trajectory class (in red), the arm is extended to the left when passing under the narrow passage while in the second class (in blue), the arm is extended to the right.

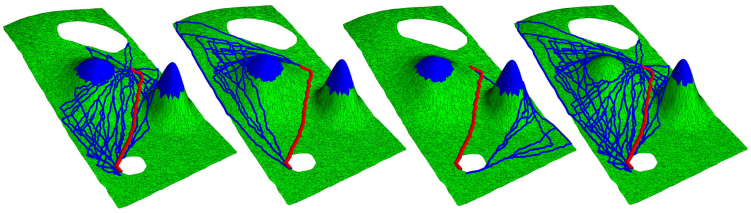


Figure 6: We display a cost function and classes of trajectories (in blue) depending on a cost threshold and a path α_0 (in red). For the higher threshold in the rightmost plot, the two classes in the two leftmost figures merge.

classes exist.

Filtrations with cost functions We consider the free configuration space $\mathcal{C}_f \subset \mathbb{R}^2$ of size 250 by 500 with two obstacles (in white) displayed in Fig. 6. We would now like to distinguish not only between homotopy classes depending on the obstacles in the configuration space, but also discern how trajectories behave with respect to the two peaks of the cost function. The simplicial complex $L_{10,\lambda}(X)$ is displayed for 10000 samples X and height values are determined by the cost function. At cost threshold $\lambda = 90$, the top of one of the hills defined by the cost-function is removed from the complex in the rightmost figure (indicated in blue), while at $\lambda = 70$ both hills are truncated in the remaining figures. We sampled 100 random paths in this configuration space by fixing the initial and terminal vertex at the start and end-point of the drawn red reference trajectory and by sampling random waypoints as before. The figure displays example trajectory classes for differing cost filtration values. Note how, at a cost threshold of $\lambda = 90$ in the right plot, the two classes depicted in the two leftmost parts of the figure merge.

Baxter robot, 3D and 6D We now investigate a kinesthetic demonstration scenario where the Baxter robot in Fig. 7 is taught a set of trajectories which we then classify topologically. In the 1^{st} experiment (E1, Fig. 7, column 1-2), the robot is shown two ways to reach from one point above its head to a point in front of its torso. Only one arm is moved in each demonstration while the other arm remains still. We used the end-effector positions of the moving arm to represent trajectories in $\mathbb{R}^3$. In the 2^{nd} (column 3-4) and 3^{rd} (column 5-6) experiment E2 and E3, we record the positions of both end-effectors during dual arm manipulations resulting in a 6D configuration space. In E2, the robot is taught to pick up a cylindrical object with both hands from a table and to move it to one of two positions, one higher and one lower than the table. We also vary the distance between the hands during grasping between demonstrations. The trajectories in E2 are periodic. The motions start with the arms in a rest position on the sides, the object is then grasped and moved, and the rest position is visited again. In experiment E3, the robot moves the same object from a horizontal to a vertical configuration, but a metal bar is located between the robot and the object. Two

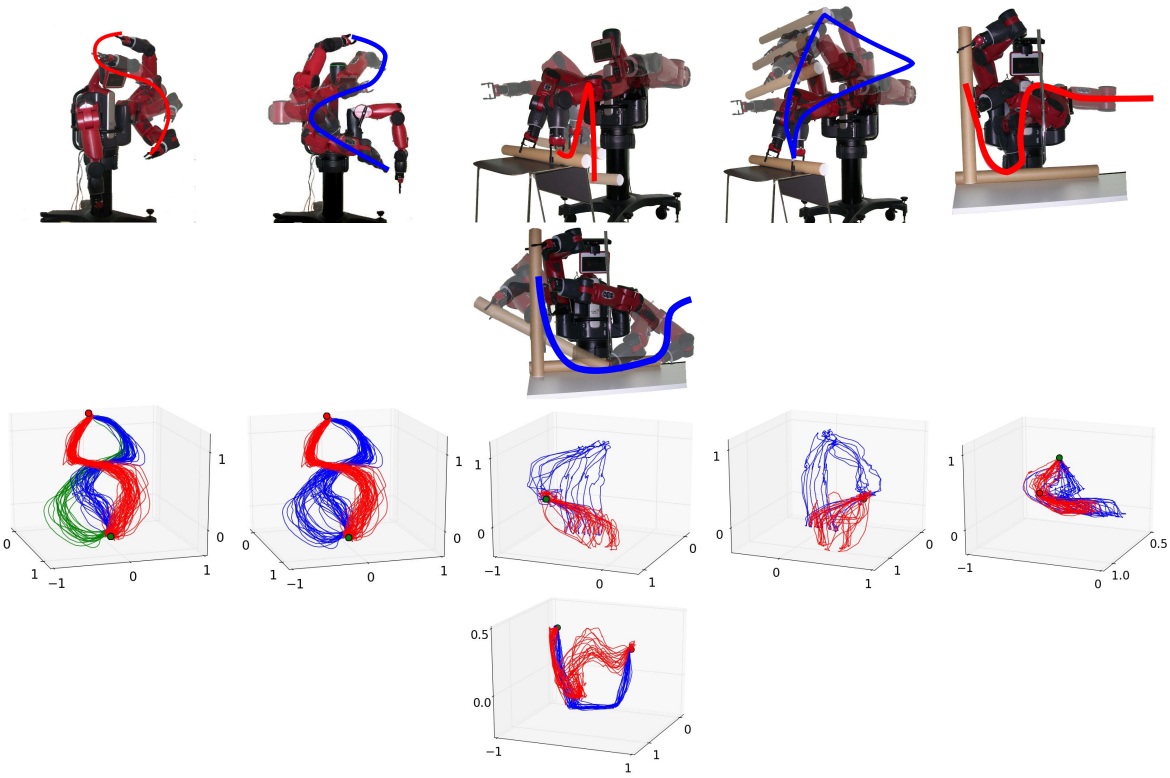


Figure 7: Baxter robot experiments (E1: column 1-2, E2: column 3-4, E3: column 5-6). Trajectory classes are illustrated in the top row and details are provided in the bottom row. We recorded 97, 18 and 40 trajectories resulting in point clouds with 9594, 3650, 4326 points in dimension 3, 6, and 6, and in filtrations with 0.19, 2.72, 3.05 million edges and triangles for E1, E2, E3 respectively. The computation times for (Delaunay triangulation, 2-skeleton and filtration computation, boundary matrix reduction) were $(0.22, 0.26, 0.12)$, $(41.84, 46.95, 44.17)$ and $(44.92, 51.99, 52.44)$ seconds for E1, E2, E3 respectively, while the classification of all trajectories in each experiment took no more than 0.05s. Bottom row: The first two images show the end-effector trajectories in E1. While we obtain 3 classes for $r = 0.08m$ (first image) the green class merges with the blue one at $r \approx 0.084m$ (second image, see Fig 8). The 3^{rd}, 4^{th} and 5^{th}, 6^{th} image show the right and left hand trajectories for E2 and E3 respectively. The trajectory classes at $r = 0.19m$ are indicated in red and blue.

intuitive motion classes are based on whether the left arm crosses in front of the obstacle, or behind it. Note that, in experiment E1 and E2, no obvious obstacles lie directly in $\mathcal{C}_f$, but due to the type of demonstrations and the robot's joint limits, the space $X_r \simeq DC_r(X) \subset \mathcal{C}_f$ exhibits interesting voids which we can exploit for classification. We found that trajectories were well-approximated using the described heuristic mapping to nearby edge-paths in $DC_R(X)$ (Sec. 2.3) for $R = 0.08m$ (experiment E1) and $R = 0.15m$ (experiment E2 and E3) respectively. For smaller r, $DC_r(X)$ was either not path-connected, or the edge path approximation deteriorated significantly. We hence investigated classifying paths in X_r for $r \geq R$. Fig. 8 displays how the number of topological trajectory classes changes with varying r, and the second row of Fig. 7 displays trajectory classes at various filtration values. In all experiments, there exists a large filtration interval with just two trajectory classes corresponding to the two intuitive classes we just described for each experiment. The first image in the second row of Fig. 7 also illustrates the three classes one obtains in E1 for a choice of $r = 0.08m$. More details on the computation times and the sizes of the simplicial complexes is provided in Fig. 7.

2.6 Future Work

We have explored a novel *sampling-based* approach to studying homotopy classes of trajectories in general configuration spaces. We believe that our approach could be incorporated with many existing algorithms.

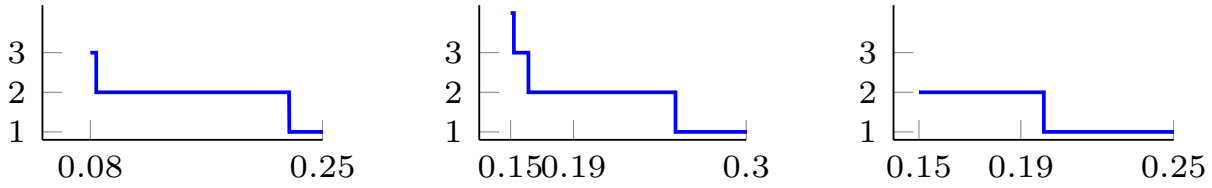


Figure 8: Number of path classes (vertical axis) vs. filtration value (horizontal axis) for experiments E1, E2, E3 from left to right. All paths exist starting at filtration value $R = 0.08m$ for E1 and $R = 0.15m$ for E2 and E3.

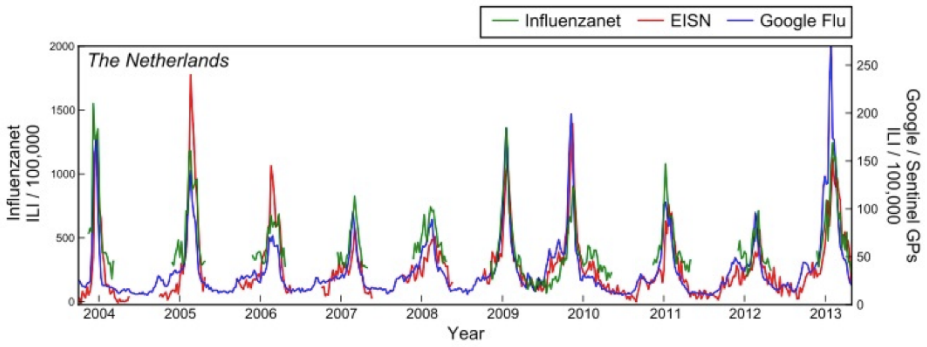


Figure 9: The comparison between the Influenzanet data with the data of both the European Influenza Surveillance Network [EISN] and the Google Flu Trends for the Netherlands in the period of 10 years of flu seasons 2003-2014. (from Influenzanet, www.influenzanet.eu)

For example, the integration of *local* trajectory optimization based algorithms with our approach which extracts *global* information about trajectories could be of interest. Another promising future application of our method could be a class-dependent generation of dynamic motion primitives. Several aspects of our approach remain to be investigated more thoroughly, such as the optimal selection of filtration parameters once the persistent cycle coordinates have been computed. While our method scaled well to large sample sets in 2 to 4 dimensions and was applicable also for a smaller set of samples in 6D, Delaunay triangulations and hence $DC_r(X)$ have an worst-case complexity of $O(n^{\lceil d/2 \rceil})$ [28] in dimension d and sample size n. In future work, we hope to investigate also alternative simplicial complexes, such as Vietoris-Rips and (weak) witness complexes [11] which approximate X_r and which could be used to mitigate the 'curse of dimensionality' in higher dimensions.

3 InfluenzaNet

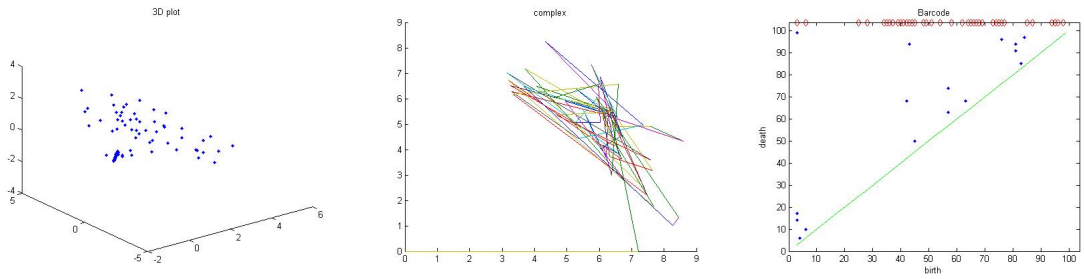
Influenzanet is a system to monitor the activity of influenza-like-illness (ILI) with the aid of volunteers via the internet. It has been operational for more than 10 years at the EU level since 2008. In contrast with the traditional system of sentinel networks of mainly primary care physicians, Influenzanet obtains its data directly from the population. This creates a fast and flexible monitoring system whose uniformity allows for direct comparison of ILI rates between countries as in [51]. Any resident of a country where Influenzanet is implemented can participate by completing an online application form, which contains various medical, geographic and behavioural questions. Participants are reminded weekly to report any symptoms they have experienced since their last visit. The incidence of ILI is determined on the basis of a uniform case definition.

Figure 10: The persistence diagram of the graphic of a real function, and correspondent barcode, capturing the topological features of the shape. The birth and death time of such features provide us the lifetimes represented in the diagram by points. (from Sketches of a Platypus)

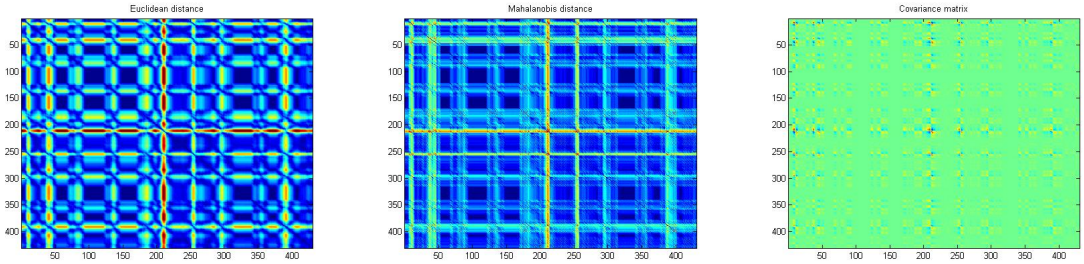
Using persistence we are able to analyze the Influenzanet data identifying several topological features relevant to the epidemiological study. In particular, we can identify data noise, distinguish higher dimension features and look at join spaces between countries. This is done both in terms of the overall structure of a disease as well as its evolution. Finally, it provides a way to test agreement at a global scale arising from standard local models.

3.1 Topological Medical Data Analysis

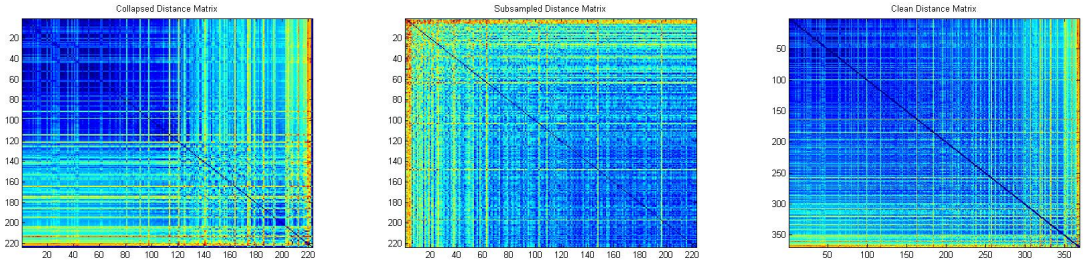
We are fundamentally interested in applying the new technology of persistence homology to have a topological analysis of the medical data provided by some of the members of the network Influenzanet. For a first approach we use the data of the partner in the Netherlands, the first one to be active in this network, collecting data from the flu season 2003/04 to the flu season 2013/14. This particular collection of data that we've been looking at presents the fields 'date', 'participants' and 'ILI' where the last field is divided in four different cases of Influenza Like Illness, according to the collected online questionaires. Below you can see the 3D plot of this data (left), the simplicial complex constructed over it (centre), and the correspondent persistence diagram.



The computation of persistence diagrams via Vietoris-Rips complexes was done using Perseus, the open source persistent homology software. Such complexes are completely determined by the underlying 1-skeleton. That structure can be represented as a symmetric distance matrix where the entries come from pairwise distances between points in a point cloud. Perseus can compute the persistent homology of Vietoris complexes from which the distance matrix was generated. The following images represent the Euclidean distance matrix (left), the Mahalanobis distance matrix (centre), and the covariance matrix for the same data (right).

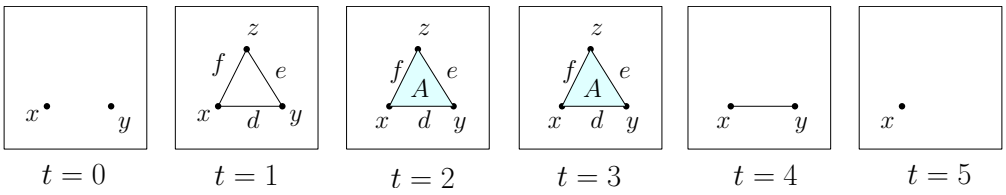


We have constructed several algorithms to clean the data prior the construction of the Vietoris-Rips complexes. The images bellow show the effect of those algorithms on the Mahalanobis metric: subsampling (left), colliding close enough data points (centre), and setting the distance of adjacent data points to zero (right).



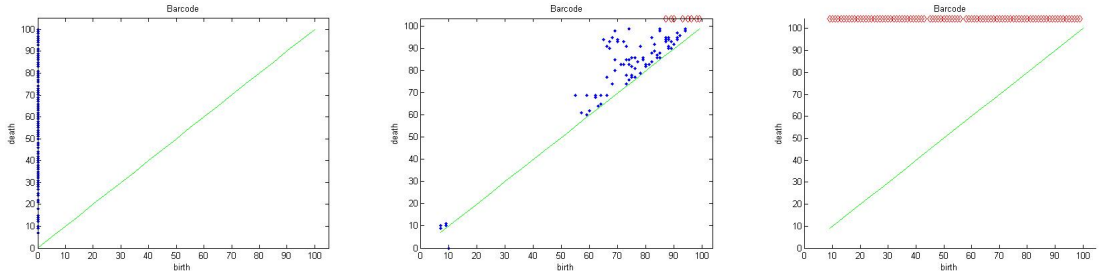
The Mahalanobis distance is a measure of the distance between a point P and a distribution D. Mahalanobis distance is widely used in cluster analysis and classification techniques.

3.2 Persistence Diagrams for Influenza

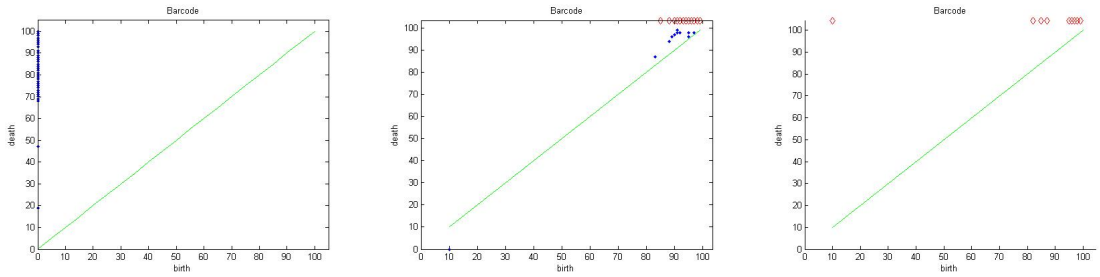


Techniques of persistence can be used to infer topological structure in data sets while certain variations on the method can be applied to study aspects of the shape of point clouds. By considering all possible scales, one can infer the correct scale at which to look at the point cloud simply by looking for scales where the persistent homology is stable. Multi-scale methods are thus enable to study the topology of point clouds as a route for approximating topological features of an unobservable geometric object generating samples. For these reasons, barcodes seen as multi-scale signatures are of great importance to the application of this research to the development of new techniques in machine learning.

These barcodes can be encoded within pairs of numbers represented in the quarter plane, indicating birth and death time of a certain topological feature of the data: persistence diagrams. Much of the applications within the recent research require manipulation and comparison of persistence diagrams. In the following figure we present the persistence diagrams for the original data regarding dimension 0 (left), dimension 1 (centre), and dimension 2 (right).



We can already see that most of the features that have a death time live in dimension 1, while dimension 2 is mostly populated by features that live forever. In the next figure we now present the persistence diagrams for the subsampled data regarding dimension 0 (left), dimension 1 (centre), and dimension 2 (right).



Future Work Topological data analysis is interested in problems relating to nonlinear systems, large scale data and development of more accurate models, that contribute to a high level research. The study of Epidemiology is a great source of problems that focus aspects of such nature. Moreover, persistence can provide such research with high dimension techniques for medical data analysis. In particular, persistence diagrams are a clear and practical tool that allows us the detection of outliers and to capture the dynamics of the system. During further research we will investigate if those exceptional points evident for the Mahalanobis metric are distinct under other metrics, and learn the appropriate metric such that those outliers are close enough. We shall also use kernels and SVMs on these features, enabling a machine learning approach over this data.

4 Topological Structures In Gene Expression Data

For the purpose of these experiments, we consider a point cloud as the input set. It is a collection of points in some (multidimensional) coordinate system. For a point cloud data we need to build a topological space for which we compute topological invariants.

Our goal is to show a topological description for each patient gene expression data. We get a new way to describe features of the data which rely on relations between transcripts. Using computational topology methods we can measure distance between the features which leads us to a new way of data clustering. Using the clusters and distances we can select patients with unusual gene expression.

4.1 Methods

Point cloud In the case of gene expression data we have on the input a matrix G with N and M columns of real numbers (N is the number of patients and M is the number of transcripts). By $G[i]$ we denote the i-th row of the matrix G for $0 \leq i < N$, i.e. a vector of M real numbers which corresponds to the i-th individual. We denote the j-th coordinate of a vector X by $X[j]$, so $G[i][j]$ for $0 \leq j < M$ is the element of G in the i-th row and the j-th column.

One can produce a point cloud data from the matrix in many different ways. First obvious point cloud data we can construct using rows of the matrix as a point in M dimensional space. In this way the point cloud represents the set of individuals. It might be good enough to compare different sets, but not to compare individuals. Our goal is to have a topological description for each individual.

Now we show a point cloud construction known from topological data analysis of time series [65]. Let X be a vector with length n. Let $1 \leq w \leq n$ be a fixed natural number, we call it a *window size*. A *point cloud* for X with window size w is denoted by $P_w(X)$ and defined as:

$$P_w(X) := \{ (X[k], X[k+1], \ldots, X[k+w]) \mid 0 \leq k < M \}, \tag{1}$$

where $X[k+w] := X[(k+w) \mod n]$ for $k+w \geq n$, i.e. we wrap around. Intuitively we group together w coordinates by moving a sliding window above the sequence X. To mesure distance between points in the point cloud we use standard euclidean metric.

4.2 Results

In this section we connect the ideas from topological data analysis (more precisely from persistent homology) with gene expression data analysis. In the Section 4.1 we show a possible way to make a point cloud data for each individual gene expression data. With persistent homology we can compute the persistence diagram for each such point cloud. Using the bottleneck distance between diagrams we can compute a distance matrix. We put together all the steps in the following algorithm:

Input: G - the gene expression matrix; w - the sliding window size.

1. For each individual i let $P[i]$ be its point cloud $P_w(G[i])$ (see (1)).
2. For each individual i let $PD[i]$ be the persistence diagram of $P[i]$.
3. For each pair (i, j) of individuals let $D[i][j]$ be the bottleneck distance between $PD[i]$ and $PD[j]$.

Output: PD - the persistence diagrams; D - the distance matrix.

The goal of the algorithm is to detect differences/similarities between individuals. Assume we consider the i-th individual. Then points in the point cloud $P[i]$ represents a group of transcripts from an individual gene expression data. If the euclidean distance between two points is less or equal to ϵ, then on the topological level we have an edge in the Vietoris-Rips complex $C_\epsilon(P[i])$. For bigger groups of points we have a triangles, tetrahedrons, and so on. It means that groups of similar regions in the gene expression data are connected by a simplices from $C_\epsilon(P[i])$. We recall that persistent homology detects holes in a topological space. It is hard to understand the meaning of multidimensional holes in a data set. For the simplicity we consider only one dimensional holes (cycles). A cycle means that there is a sequence of points $p_1, p_2, \ldots, p_n, p_1$ from the point cloud data, where the distance between adjacent points is less or equal to ϵ, but its diameter is greater than ϵ. For the gene expression data each cycle means that there is a set of regions (each build on w transcripts) with periodic behavior. On the other side persistent diagrams visualize lifespan for holes. Statistically it is almost impossible to get long living holes generated by noisy or sampling error, they must be important features of the data.

From the above observations we make a hypothesis that a persistence diagram encodes features of an individual gene expression data. We executed the algorithm for all individuals and various values of the parameter w. On the Figure 11 and the Figure 12 we present persistence diagrams for individuals T2DG0500351 and T2DG1600797. Points faraway from the diagonal represent persistent holes - with long lifespan.. The two diagrams realize the maximum of matrix D computed by the algorithm with $w = 32$. For the same parameter w on the Figure 13 is the distance matrix D reordered according to hierarchical clustering.

As a comparison we computed distances between individuals using rows of the matrix G and euclidean metric (full transcript of gene expressions is a feature vector for an individual). On the Figure 14 is the distance matrix reordered using hierarchical clustering. We observe that the topological approach gives more

clusters. Also we can see that the clusters are more accurate, i.e. the difference between clusters is more visible.

Figure 11: Persistence diagram for individual T2DG0500351.

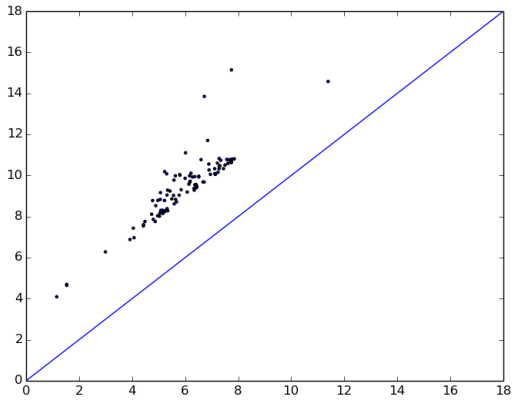


Figure 12: Persistence diagram for individual T2DG1600797.

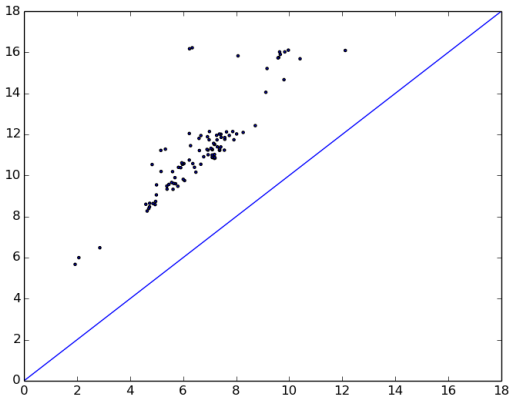


Figure 13: Distance matrix of the individuals persistence diagrams with hierarchical clustering.

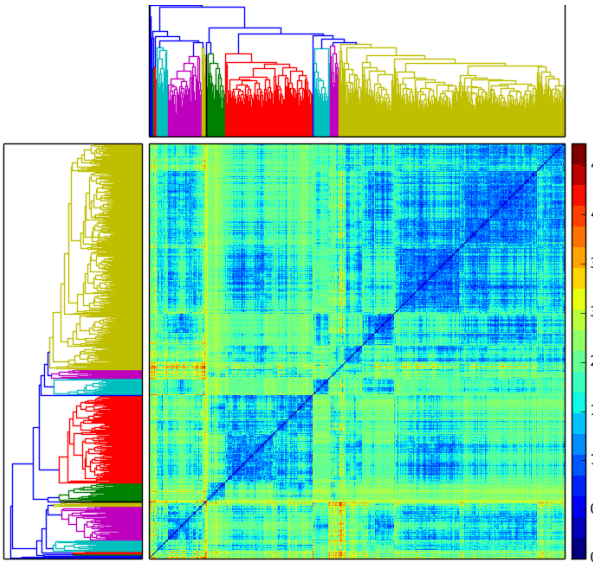
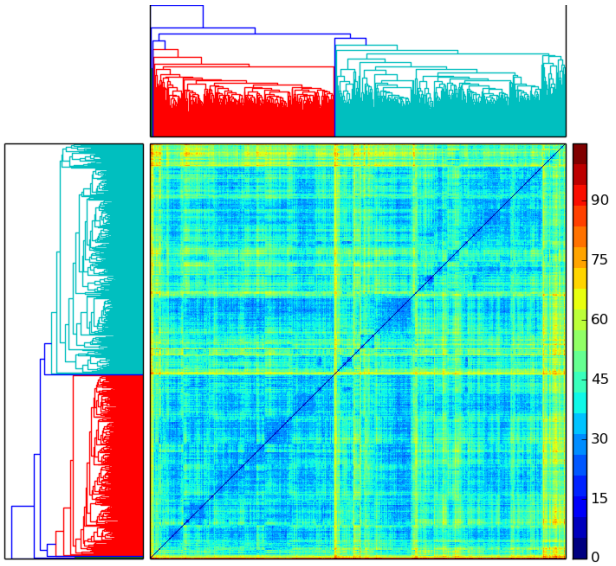


Figure 14: Distance matrix of the original gene expression data set with hierarchical clustering.



4.3 Discussion

In this study we propose topological data analysis techniques to analyze gene expression data sets. We describe one of many possible ways to convert raw data sets to point clouds, which are natural inputs for persistent homology algorithms. Significant difference between individuals visible on the persistence diagrams suggest an important information hidden in the data and possibly not visible by other approaches. We claim that the diagrams carry the essence of multidimensional data. We can treat it as a new way of feature extractions. In that way we can try to investigate a corelation of the topological features with other properties of the individuals. It requires more work with well described and measured data sets. We tried to use the distance matrix with cassification algorithms for blood presure and family data - without success. Our intuition suggest that a knowledge about the data is in the topological features.

5 Distributed Computation of Persistence

On the practical side of the computation of persistence, it has been observed that the standard algorithm usually performs much better on real-world instances than predicted by the worst-case bounds, and relatively simple optimizations of the standard method yield remarkable speed-ups [37]. Recently, Maria et al. [44] implemented a memory-efficient and comparably fast method for computing persistent cohomology, which yields the same information about birth and death of features as its homology counterpart. Further improvements have been reported by using several processors in a shared memory environment [34] (see also [42, 43] for alternative parallelization schemes). With these optimizations, it is often the case that computing persistence actually takes less time than even reading the input into memory. Therefore, the limiting factor is not so much the time spent for computation but rather the memory available on the computer.

Contribution We present a scalable algorithm for computing persistent homology in parallel in a distributed memory environment. This method the computation of much larger instances than using existing state-of-the art algorithms on a single machine, by using sufficiently many computing nodes such that the data fits into the distributed memory. While overcoming the memory bottleneck is the primary purpose of our approach, we aim for a time-efficient solution at the same time.

As demonstrated by our experiments, our implementation exhibits excellent scaling with respect to memory usage on a single node, and even outperforms similar parallel shared memory code in running time. This result is somewhat surprising, since the computation of topological properties like persistent homology is a global problem, and at first sight it is not obvious at all that the computation can be performed with the very simple and inexpensive pattern of communication that our algorithm exhibits.

Our method closely resembles the *spectral sequence algorithm* for persistent homology [39, S VII.4]. However, several adaptions are necessary for an efficient implementation in distributed memory. Most importantly, reduced columns are not stored in order of their index in the matrix, but rather according to the order of their *pivot*, the largest index of a non-zero entry. This allows a node to perform eliminations in its associated rows, and to determine if a column with pivot in these rows is reduced, without further communication with other nodes. Furthermore, we minimize the number of messages sent through the network by collecting blocks of messages, and we simplify the communication structure by letting node j only communicate with nodes $j \pm 1$. Finally, we incorporate the clear optimization [37] into the algorithm in order to avoid unnecessary column operations.

Boundary matrix For a matrix $M \in (\mathbb{Z}_2)^{n\times n}$, let M_j denote its j^{th} column and $M_j^i \in \mathbb{Z}_2$ its entry in row i and column j. For a column M_j, we define $\operatorname{pivot}(M_j) = \min\{p \in \mathbb{N}_0 : M_j^i = 0 \text{ for all } i > p\}$ and call it the *pivot index* of that column. When obvious from the context, we omit explicit mention of the matrix M and write $\operatorname{pivot}(j)$ for $\operatorname{pivot}(M_j)$.

The *boundary matrix* $D \in (\mathbb{Z}_2)^{n\times n}$ of a simplexwise filtration $(K_i)_{i=1}^n$ is the $n \times n$ matrix of the boundary operator $\partial_* : C_*(K) \to C_*(K)$ with respect to the ordered basis $(\sigma_i)_{i=1}^n$ of $C_*(K)$. We have $D_j^i = 1$ if and

only if σ_i is a face of σ_j of codimension 1. In other words, the j^{th} column of D encodes the boundary of σ_j. D is an upper-triangular matrix, since any face of σ_j must precede σ_j in the filtration.

Matrix reduction A column operation of the form $M_j \leftarrow M_j + M_k$ is called *left-to-right addition* if $k < j$. A left-to-right addition is called *eliminating* if it decreases pivot(j). A column M_j is called *reduced* if pivot(j) cannot be decreased by applying any sequence of left-to-right additions. In particular, there is no non-zero column M_k with $k < j$ and pivot$(k) =$ pivot(j). Clearly a zero column is reduced. Note that a reduced column remains reduced under eliminating left-to-right column additions.

We call a matrix M *reduced* if all columns are reduced, or equivalently, if no two non-zero columns have the same pivot index. We call M *reduced at index* (i, j) if the lower left submatrix of M with rows of index $> i$ and columns of index $\leq j$ is reduced. A sufficient condition for column M_j to be reduced is that M is reduced at index (i, j) with $i =$ pivot(j).

If R is a reduced matrix obtained by applying left-to-right additions to M, we call it a *reduction of* M. In this case, we define

$$P_R := \{(i, j) \mid i = \text{pivot}(R_j) > 0\}$$

Although the reduction matrix R is not unique, the set P_R is the same for any reduction of M; therefore, we can define P_M to be equal to P_R for any reduction R of M.

Persistence by reduction For the boundary matrix D of the filtration $(K_i)_{i=1}^n$, the first i columns generate the boundary group $B_*(K_i)$. This property is invariant under left-to-right column additions. For a reduction R of D, the non-zero columns among the first i columns actually form a basis of $B_*(K_i)$. Note that

$$\begin{aligned} i = \dim C_*(K_i) &= \dim Z_*(K_i) + \dim B_*(K_i) \text{ and} \\ \dim H_*(K_i) &= \dim Z_*(K_i) - \dim B_*(K_i). \end{aligned}$$

Hence, if R_i is zero, we have

$$\begin{aligned} \dim B_*(K_i) &= \dim B_*(K_{i-1}), \\ \dim Z_*(K_i) &= \dim Z_*(K_{i-1}) + 1, \text{ and} \\ \dim H_*(K_i) &= \dim H_*(K_{i-1}) + 1, \end{aligned}$$

and so some homology class is born at i. If on the other hand R_j is non-zero with $i =$ pivot(j), we have

$$\begin{aligned} \dim B_*(K_i) &= \dim B_*(K_{i-1}) + 1, \\ \dim Z_*(K_i) &= \dim Z_*(K_{i-1}), \text{ and} \\ \dim H_*(K_i) &= \dim H_*(K_{i-1}) - 1. \end{aligned}$$

The fact that R_j has pivot i means that $R_j \in Z_*(K_i)$ and hence

$$[R_j] \in H_*(K_i);$$

the fact that it is reduced means that there is no $b \in B_*(K_{j-1})$ with $b + R_j \in Z_*(K_{i-1})$ and hence

$$[R_j] \notin \text{im} h_{i-1}^i.$$

We conclude that $[R_j]$ is born at i. We even have

$$[R_j] \notin \text{im} h_{i-1}^{j-1}.$$

Moreover, R_j is a boundary in K_{j-1}, and so

$$[R_j] = 0 \in \text{im} h_{i-1}^j.$$

We conclude that the pairs $(i, j) \in P_D$ are the persistence pairs of the filtration.

The standard way to reduce D is to process columns from left to right; for every column, previously reduced columns are added from the left until the pivot index is unique. A lookup table can be used to identify the next column to be added in constant time. The running time is at most cubic in n, and this bound is actually tight for certain input filtrations, as demonstrated in [46].

Clearing optimization Despite its worst-case behavior, there are techniques to speed up the reduction significantly in practice. A particularly simple yet powerful improvement has been presented in [37]. It is based on the following observations.

First, the reduction of the matrix can be performed separately for each dimension d, by restricting to the submatrix corresponding to columns of dimension d and rows of dimension $d-1$. This submatrix is exactly the matrix of the boundary operator $\partial_d : C_p(K) \to C_{p-1}(K)$. The second basic fact to note is that in any reduction of D, if i is a pivot of some column j, the i^{th} column is zero.

This leads to the following variant of the reduction algorithm: the boundary matrix is reduced separately in each dimension in decreasing order. After the reduction in dimension d, all columns corresponding to pivots indices are set to zero – we call this process *clearing*. Note that columns corresponding to d-simplices have pivots corresponding to $d-1$-simplices. After clearing, we proceed with the reduction in dimension $d-1$.

5.1 Algorithm

Throughout the section, let $(K_i)_{i=1}^n$ be a filtration of a simplicial complex consisting of n simplices, represented by its boundary matrix D. Our goal is to compute the persistence pairs of $(K_i)_i$ on a cluster of p processor units, called *nodes*, which are indexed by the integers $\{1, \dots, p\}$.

Reduction in blocks Let $0 = r_0 < \cdots < r_i < \cdots < r_p = n$ be an integer partition of the interval $\{0, \dots, n\}$. Let the i^{th} *range* be the interval of integers k with $r_{i-1} < k \leq r_i$. We define the *block* (i, j) of M as the block submatrix with rows from the i^{th} row range and columns from the j^{th} columns range. The blocks partition the matrix into p^2 submatrices. Any block (i, j) with $i > j$ is completely zero, since D is lower triangular.

To simplify notation, we call M *reduced at block* (i, j) if M is reduced at index (r_{i-1}, r_j). Moreover, we call M *reducible in block* (i, j) if M is reduced at block $(i, j-1)$ and at block $(i+1, j)$. This terminology is motivated by the fact that in order to obtain a matrix that is reduced at block (i, j), only entries in block (i, j) have to be eliminated, as described in 1 and shown in the following lemma.

Algorithm 1 Block reduction

Require: input M is reducible in block (i, j)
Ensure: result M is reduced at block (i, j)

```
1: procedure ReduceBlock(i, j)
2:     for each l in range j in increasing order do
3:         while ∃k with pivot(k) = pivot(l) in range i do
4:             add column k to column l
5:         end while
6:         if pivot(l) is in range i then
7:             add column l to collection of reduced columns
8:         end if
9:     end for
10: end procedure
```

Lemma 2. *1 is correct: if M is reducible in block (i, j), then applying 1 yields a matrix which is reduced at block (i, j).*

Proof. By induction on l, M is reduced at index (r_{i-1}, l) after each iteration of the main **for** loop (2). This follows directly from the exit condition of the **while** loop in 3, together with the induction hypothesis and the precondition that M is reduced at index (r_i, r_j) and hence also at index (r_i, l). □

Lemma 3. *1 only requires access to the unreduced columns of M in range j and the reduced columns with pivot in range i.*

Proof. Let l be in range j and let $k < l$ be such that $\mathrm{pivot}(k) = \mathrm{pivot}(l)$ is in range i, as in the **while** loop in 3. Then clearly column l is unreduced. Moreover, as shown in the proof of 2, M is reduced at index (r_{i-1}, l). Since $\mathrm{pivot}(k)$ is in range i, we have $r_{i-1} < \mathrm{pivot}(k)$, and by assumption $k < l$. Hence M is also reduced at index $(\mathrm{pivot}(k), k)$, i.e., column k is reduced. □

Parallel reduction We now describe a parallel algorithm to reduce a boundary matrix D by applying block reduction on all blocks (i, j) with $i \leq j$ in a certain order.

The algorithm reduces the blocks starting with the diagonal blocks (i, i) with $1 \leq i \leq p$. Indeed, note that the boundary matrix D is (i, i)-reducible for any diagonal block (i, i). All block reductions for diagonal blocks are independent and can be performed in parallel. Now consider a block of the form (i, j) with $i < j$. Note that this block can be reduced as soon as blocks the $(i, j-1)$ and $(i-1, j)$ have been reduced. This relation defines a partial order on the blocks (i, j) with $i \leq j$. If the order of execution of the block reductions is consistent with that partial order, the preconditions of block reduction are satisfied in every block. Note that two blocks (i, j) and (i', j') can be reduced independently iff either ($i < i'$ and $j < j'$) or ($i > i'$ and $j > j'$). After having reduced the block $(1, p)$, the postcondition of 1 yields that the resulting matrix is a reduction of the input boundary matrix D.

Note that a special case of this block-reduction scheme is the *spectral sequence algorithm* presented in [39, S VII.4]. This algorithm sweeps the blocks diagonally, and in each phase $r \in \{1, \ldots, p\}$ of the sweep it reduces all blocks (i, j) with $j - i = r - 1$ in order of increasing index i. The algorithm as described is sequential, however, as discussed above, within a given phase r the blocks can be reduced independently.

Distributed reduction We now describe how the data and the workload are distributed and transferred between the nodes.

Each node i is assigned a row of blocks for reduction. The blocks are necessarily processed from left to right. Recall that reducing a block (i, j) requires access to the unreduced columns in range j, and to the reduced columns with pivot in range i. During the execution, each node i maintains a collection of all reduced columns with pivot in the i^{th} range, indexed by pivot. The unreduced columns in a given range j, on the other hand, are passed on from node to node. No data is duplicated among the nodes; each column of the matrix is stored in exactly one node throughout the execution of the algorithm. The union of the locally stored unreduced and reduced columns yields a distributed representation of the partially reduced boundary matrix.

Initially, each node i loads the columns of the input boundary matrix in range i. The following procedure is now repeated, with j ranging from i to m. Node i performs reduction in block (i, j) and retains the reduced columns with pivot in range i in its collection. After that, it sends a package to node $i - 1$ containing the remaining unreduced columns in range j (if $i > 1$), and receives a package from node $i + 1$ containing the unreduced columns in range $j + 1$ (if $j < p$).

Observe that in each iteration, node i has all the information required to perform reduction in block (i, j), namely, the unreduced columns in range j and the reduced columns with pivot in range i. Moreover, the preconditions for block reduction are satisfied, since block $(i, j-1)$ is reduced on the same node i before block (i, j), and block $(i+1, j)$ is reduced on node $i + 1$ before node i receives the unreduced columns in range j from node $i + 1$. We conclude:

Lemma 4. *If 2 is executed on a cluster with p nodes, it computes a reduction of the input matrix.*

Note that the structure of communication between the nodes is very simple: each node i only receives data from node $i + 1$ and only sends data to node $i - 1$. Moreover, less than p messages are sent between

Algorithm 2 Distributed matrix reduction

Require: access to columns of input boundary matrix D in range j
Ensure: resulting output matrix R is a reduction of D

```
1: procedure ReduceOnNode(i)
2:     input package with columns of D in range i
3:     for j = i, ..., p do
4:         ReduceBlock(i, j)
5:         if i > 1 then
6:             send package with unreduced columns in range j to node i − 1
7:         end if
8:         if j < p then
9:             receive package with unreduced columns in range j + 1 from node i + 1
10:        end if
11:    end for
12:    return reduced columns with pivot in range i
13: end procedure
```

each pair of consecutive nodes. This is highly beneficial for distributed computing, as the communication overhead and the network latency become negligible.

Clearing in parallel The clearing optimization from Section 2.2 can be implemented in the distributed reduction algorithm with minor changes. Recall that the the clearing optimization iterates over the dimensions d in decreasing order and processes only the columns of a given dimension d at a time.

The ranges are defined by a single global partition $r_0 < \ldots < r_p$ that does not change per dimension. Note that this might cause initial column packages of different sizes in a given dimension, even if the ranges are all of same size. However, it has the following advantage: when node i has performed its last block reduction for dimension d, it knows all pivots that fall in the i^{th} range. All these pivots corresponds to $d-1$-simplices that create homology and hence correspond to zero columns in any reduction. In the next iteration, node i is initialized to process the columns of dimension $d-1$ in the i^{th} range. Before it starts the block reduction, it can simply clear all columns with indices that were pivots in dimension d. In particular, no communication with other nodes is required.

Design rationale We justify some design choices in our algorithm and discuss alternatives. First, we implemented the sending of packages in Algorithm 2 in a *blocking* fashion, i.e., a node does not start receiving the next package until has sent and discarded its current package. Clearly, this strategy can result in delayed processing of packages because a sending node has to wait for its predecessor to be ready to receive a package. On the other hand, the strategy guarantees that every node holds at most one package at a time; this prevents a slower node from accumulating more and more packages, possibly causing high memory consumption.

A possible strategy to reduce the overall amount of communication would be to have node i send a unreduced column with pivot in the k^{th} range to node k directly, instead of the predecessor node $i-1$. However, this approach would complicate the communication structure and data management significantly. Any node would have to be able to receive unreduced columns any time, and it would not be possible to bound the number of unprocessed columns a node has to maintain in memory. It would also increase the number of messages send through the network.

A somewhat dual approach to our communication scheme would be to send the reduced columns from node i to $i+1$ instead of sending the unreduced columns from node i to $i-1$. In this variant, node i would perform reduction in block (j,i) for $j = i, i-1, \ldots, 1$. However, in this approach, the package size would increase towards the end of the reduction, as the number of reduced columns increases, whereas in our implementation the package size decreases together with the number of reduced columns. Since typically

	PHAT		DIPHA				
cores/nodes	1	16	2	4	8	16	32
GRF2-256	10.2GB	10.5GB	11.1GB	5.6GB	2.8GB	1.4GB	0.74GB
GRF1-256	10.8GB	11.3GB	11.8GB	6.1GB	3.1GB	1.5GB	0.8GB
GRF2-512						11.1GB	5.7GB
GRF1-512							9.1GB
vertebra16							9.0GB

Table 1: Peak memory consumption for sequential and parallel shared memory (PHAT, left) algorithms and our distributed algorithm (DIPHA, right)

	PHAT		DIPHA				
cores/nodes	1	16	2	4	8	16	32
GRF2-256	14.6s	5.2s	10.1s	5.5s	3.4s	2.2s	1.6s
GRF1-256	28.8s	12.8s	27.2	20.3	15.4	12.1s	9.9s
GRF2-512						17.9s	11.2s
GRF1-512							95.3s
vertebra16							34.9s

Table 2: Running times for sequential and parallel shared memory (PHAT, left) algorithms and our distributed algorithm (DIPHA, right)

most columns are reduced early on, we expect much more data to be sent between the nodes using this variant.

5.2 Experiments

Since our algorithm is, to the best of our knowledge, the first attempt at computing persistence in a distributed memory context, we concentrate our experimental evaluation on two aspects. First, how does our approach scale with an increasing number of nodes, in running time and memory consumption? Second, how does the our algorithm compare with state-of-the-art sequential and parallel shared memory implementations on instances which are still computable in this context?

We implemented Algorithm 2 in C++ using the OpenMPI implementation of the Message Parsing Interface standard(www.open-mpi.org). We ran the distributed algorithm on a cluster with up to 32 nodes, each with two Intel Xeon CPU E5-2670 2.60GHz processors (8 cores each) and 64GB RAM, connected by a 40Gbit Infiniband interconnect.

For comparison, our tests also include results for the PHAT library(http://phat.googlecode.com), which contains efficient sequential and parallel shared memory algorithms for computing persistence. Among the sequential versions, the `--twist` algorithm option, which is the standard reduction with the clearing optimization described in Section 2.2, together with the `--bit_tree_pivot_column` data structure option, showed the overall best performance (see the PHAT documentation for more information). For parallel shared memory, the `--block_spectral_sequence` algorithm with the `--bit_tree_pivot_column` data structure showed the overall best performance on the tested examples. We therefore used these two variants for comparison. The sequential and parallel shared memory algorithms were run on a single machine of the cluster. In order to obtain a clear comparison between the shared memory and distributed memory algorithms, in our test of the distributed algorithm only one processor core per node was used.

For our tests, we focus on filtrations induced by 3D image data. In particular, we used isotropic Gaussian random fields whose power spectral density is given by a power law $\|x\|^{-p}$. This process is commonly used in physical cosmology as a model for cosmic microwave background [47]. We consider two images sizes: filtrations of images of size 256^3 have a length of $n = 511^3 \approx 133$ millions and a binary file size of around 5GB, while images of size 512^3 yield a filtration of length $n = 1023^3 \approx 1.07$ billions and a file size of around 40GB. In addition, we included the 512^3 medical image `vertebra16` from the VOLVIS repository(Available

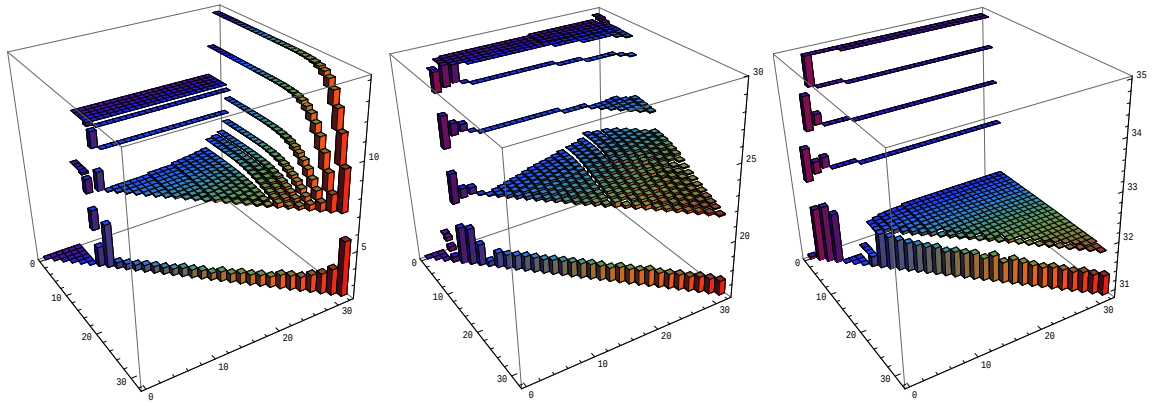


Figure 15: Running times for each block reduction in dimensions $\delta = 3, 2, 1$ for the vertebra16 data set.

at http://volvis.org) in our test set, a rotational angiography scan of a head with an aneurysm.

Scalability Tables 1 and2 show the running time and peak memory consumption of our algorithm for images of size 256^3 and 512^3. The table is incomplete; the algorithm was not able to compute a result for the remaining cases because of address space limitations of the OpenMPI I/O API that we plan to circumvent in a forthcoming version. We observe that the memory usage per node is almost exactly halved when doubling the number of nodes. For the running time, the speed-up factor is not quite as high, but still the algorithm terminates faster when using more nodes. In summary, this provides strong evidence that our algorithm scales well with the number of nodes, both regarding time and space complexity.

Comparison Tables 1 and 2 also lists the results for the best sequential and parallel shared memory algorithms of the PHAT library. Both algorithms run out of memory when trying to compute persistence for larger examples on our testing machine, showing that our distributed approach indeed extends the set of feasible instances. Moreover, we observe that the running time on 16 nodes with distributed memory is actually lower than that of the parallel shared memory algorithm on a single machine with 16 processor cores. One reason might be that the distributed system has a much larger total amount of processor cache available than the shared memory system. Since matrix reduction is more memory intensive than processor intensive, this effect may actually outweigh the overhead of communication over the network. This suggests that the distributed approach may be preferable even if the solution is in principle computable in a non-distributed environment.

Communication analysis We give more details on the amount of data transmitted between the nodes by our algorithm. Table 3 shows the total amount of data exchanged; 4 shows the largest total amount of data transmitted between any pair of nodes; 5 shows the largest package size. For the more challenging examples, the amount is in the range of GBs. Considering the bandwidth of modern interconnects and the fact that communication is bundled in a small number of packages, the running time of the local block reductions dominates the time spent for communication. This is illustrated in 15, which shows a plot of the running times for each block reduction for the vertebra16 data set on 32 nodes.

5.3 Conclusion

We presented the first implementation of an algorithm for computing persistent homology in a distributed memory environment. While our algorithm resembles the spectral sequence algorithm for persistence computation to a large extent, several lower-level design choices were necessary for an efficient realization. Our

nodes	2	4	8	16	32
GRF2-256	5.6MB	15.1MB	32.5MB	67.7MB	136MB
GRF1-256	69.2MB	218MB	497MB	1.0GB	2.0GB
GRF2-512				342MB	694MB
GRF1-512					34.0GB
vertebra16					19.1GB

Table 3: Total size of all packages sent over the network

nodes	2	4	8	16	32
GRF2-256	5.6MB	5.6MB	5.6MB	6.5MB	8.7MB
GRF1-256	69.2MB	90.3MB	109MB	162MB	238MB
GRF2-512				29.2MB	29.7MB
GRF1-512					5.0GB
vertebra16					4.2GB

Table 4: Maximum total size of all packages transmitted between any pair of nodes

nodes	2	4	8	16	32
GRF2-256	3.1MB	2.9MB	2.7MB	3.6MB	2.5MB
GRF1-256	61.4MB	52.0MB	69.0MB	50.9MB	38.4MB
GRF2-512				11.5MB	9.0MB
GRF1-512					1.9GB
vertebra16					1.5GB

Table 5: Maximum package size sent over the network

approach permits the computation of instances that were infeasible for previous methods, and the parallelism also speeds up the computation for previously feasible instances.

We plan to extend our experimental evaluation in future work. One problem in benchmarking our new approach is that persistence computation is only the second step in the pipeline: first, one has to generate a filtration that serves as the input for the algorithm. This itself usually requires a massive computation, which at some point becomes infeasible on single machines as well. We are currently working on methods for generating filtrations of large 3D images and Rips filtrations in a distributed memory environment.

References

[1] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

[2] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Software*, 22(4), 1996.

[3] U. Bauer and H. Edelsbrunner. The morse theory of Čech and Delaunay filtrations. In *Proc. of the Thirtieth Annual Symp. on Comp. Geometry*, SOCG'14, pages 484:484–484:490, New York, NY, USA, 2014. ACM.

[4] U. Bauer, M. Kerber, and J. Reininghaus. PHAT (Persistent Homology Algorithm Toolbox). http://code.google.com/p/phat/.

[5] S. Bhattacharya, V. Kumar, and M. Likhachev. Search-based path planning with homotopy class constraints. In *Proc. of The Twenty-Fourth AAAI Conf. on Artificial Intelligence*, Atlanta, Georgia, 11-15 July 2010.

[6] S. Bhattacharya, M. Likhachev, and V. Kumar. Identification and representation of homotopy classes of trajectories for search-based path planning in 3D. In *Proc. of Robotics: Science and Systems*, 27-30 June 2011.

[7] S. Bhattacharya, D. Lipsky, R. Ghrist, and V. Kumar. Invariants for homology classes with application to optimal search and planning problem in robotics. *Electronic pre-print*, Aug 2012. arXiv:1208.0573.

[8] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In *Springer handbook of robotics*, pages 1371–1394. Springer, 2008.

[9] O. Brock and O. Khatib. Real-time re-planning in high-dimensional configuration spaces using sets of homotopic paths. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'00), 2000.*, volume 1, pages 550–555. IEEE, 2000.

[10] J. Canny. *The complexity of robot motion planning*. MIT press, 1988.

[11] G. Carlsson. Topology and data. *Bull. Amer. Math. Soc. (N.S.)*, 46(2):255–308, 2009.

[12] C. Chen and M. Kerber. Persistent homology computation with a twist. In *Proc. of the 27th European Workshop on Computational Geometry*, 2011.

[13] R. Diankov and J. Kuffner. OpenRAVE: A Planning Architecture for Autonomous Robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, Pittsburgh, PA, July 2008.

[14] H. Edelsbrunner. The union of balls and its dual shape. *Discrete and Comp. Geometry*, 13(1):415–440, 1995.

[15] H. Edelsbrunner and J. Harer. Persistent homology-a survey. *Contemporary mathematics*, 453:257–282, 2008.

[16] H. Edelsbrunner and J. L. Harer. *Computational topology: an introduction*. AMS Bookstore, 2010.

[17] L. Jaillet and T. Siméon. Path deformation roadmaps: Compact graphs with useful cycles for motion planning. *Int. Journal of Robotics Research*, 27(11-12):1175–1188, 2008

[18] L. Jaillet and T. Siméon. Path deformation roadmaps. In S. Akella, N. M. Amato, W. H. Huang, and B. Mishra, editors, *Algorithmic Foundation of Robotics VII*, volume 47 of *Springer Tracts in Advanced Robotics*, pages 19–34. Springer, 2008.

[19] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. Journal of Robotics Research*, 30(7):–894, 2011.

[20] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation*, 12(4): 566–580, 1996.

[21] S. Kim, K. Sreenath, S. Bhattacharya, and V. Kumar. Optimal trajectory generation under homology class constraints. In *51st IEEE Conf. on Decision and Control*, 10-13 Dec 2012.

[22] R. A. Knepper, S. S. Srinivasa, and M. T. Mason. Toward a deeper understanding of motion alternatives via an equivalence relation on local paths. *Int. Journal of Robotics Research*, 31(2):167–186, 2012.

[23] J.-C. Latombe. *Robot Motion Planning*. Springer, 1991.

[24] S. M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

[25] S. M. LaValle and J. J. Kuffner. Rapidly-Exploring Random Trees: Progress and Prospects. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308, Wellesley, MA, 2001. A K Peters.

[26] S. R. Lindemann and S. M. LaValle. Current issues in sampling-based motion planning. In *Robotics Research*, pages 36–54. Springer, 2005.

[27] E. Masehian and D. Sedighizadeh. Classic and heuristic approaches in robot motion planning - a chronological review. *World Academy of Science, Engineering and Technology*, 23:101–106, 2007.

[28] P. McMullen. The maximum numbers of faces of a convex polytope. *Mathematika*, 17(02):179–184, 1970.

[29] B. Morris and M. Trivedi. Learning trajectory patterns by clustering: Experimental studies and comparative evaluation. In *IEEE Int. Conf. on Comp. Vision and Pattern Recognition (CVPR'09)*, pages 312–319. IEEE, 2009.

[30] P. Niyogi, S. Smale, and S. Weinberger. Finding the homology of submanifolds with high confidence from random samples. *Discrete and Comp. Geometry*, 39(1-3):419–441, 2008.

[31] J. T. Schwartz and M. Sharir. On the piano movers problem. II. General techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4(3):–351, 1983.

[32] L. Zhang, Y. J. Kim, and D. Manocha. A hybrid approach for complete motion planning. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, (IROS'07)*, pages 7–14. IEEE, 2007.

[33] Y. Zheng and X. Zhou. *Computing with spatial trajectories*. Springer, 2011.

[34] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Clear and compress: Computing persistent homology in chunks. In *TopoInVis 2013*, 2013.

[35] F. Chazal, D. Cohen-Steiner, M. Glisse, L. Guibas, and S. Oudot. Proximity of persistence modules and their diagrams. In *Proc. 25th ACM Symp. on Comp. Geom.*, pages 237–246, 2009.

[36] Chao Chen and Michael Kerber. An output-sensitive algorithm for persistent homology. In *Proceedings of the 27th Annual Symposium on Computational Geometry*, pages 207–215, 2011.

[37] Chao Chen and Michael Kerber. Persistent homology computation with a twist. In *27th European Workshop on Computational Geometry (EuroCG)*, pages 197–200, 2011. Extended abstract.

[38] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Discrete and Computational Geometry*, 37:103–120, 2007.

[39] H. Edelsbrunner and J. Harer. *Computational Topology, An Introduction*. American Mathematical Society, 2010.

[40] Herbert Edelsbrunner and John Harer. Persistent homology a survey. In Jacob E. Goodman, János Pach, and Richard Pollack, editors, *Surveys on Discrete and Computational Geometry: Twenty Years Later*, Contemporary Mathematics, pages 257–282. 2008.

[41] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. *Discrete and Computational Geometry*, 28:511–533, 2002.

[42] R. H. Lewis and A. Zomorodian. Multicore homology. Manuscript, 2012.

[43] David Lipsky, Primoz Skraba, and Mikael Vejdemo-Johansson. A spectral sequence for parallelized persistence. arXiv:1112.1245, 2011.

[44] Clement Maria, Jean-Daniel Boissonnat, and Tamal Dey. The compressed annotation matrix: An efficient data structure for computing persistent cohomology. In *ESA 2013*, 2013.

[45] Nikola Milosavljević, Dmitriy Morozov, and Primož Škraba. Zigzag persistent homology in matrix multiplication time. In *Proceedings of the 27th Annual Symposium on Computational Geometry*, pages 216–225, 2011.

[46] Dmitriy Morozov. Persistence algorithm takes cubic time in the worst case. In *BioGeometry News*. Duke Computer Science, Durham, NC, 2005.

[47] John Peacock. *Cosmological Physics*. Cambridge University Press, 1999.

[48] Afra Zomorodian and Gunnar Carlsson. Computing persistent homology. *Discrete and Computational Geometry*, 33:249–274, 2005.

[49] G. Carlsson. Topology and data. *American Mathematical Society*, 46(2):255–308, 2009.

[50] G. Carlsson and A. Zomorodian. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005.

[51] D. Paolotti A. Carnahan, V. Colizza, K. Eames, J. Edmunds, G. Gomes, C. Koppeschaar, M. Rehn, R. Smallenburg, C. Turbelin, S. Van Noort, and A. Vespignani. Web-based participatory surveillance of infectious diseases: the influenzanet participatory surveillance experience. *Clinical Microbiology and Infection*, 20(1):17–21, 2014.

[52] J. M. Chan, Joseph Minhow, Gunnar Carlsson, and Raul Rabadan. Topology of viral evolution. *Proceedings of the National Academy of Sciences*, 110(46):18566–18571, 2013.

[53] Joao Pita Costa and Primož Škraba. A topological data analysis approach to epidemiology. *Manuscript*, 2014.

[54] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete and Computational Geometry*, 28(4):511–533, 2002.

[55] Robert Ghrist. Barcodes: the persistent topology of data. *Bulletin of the American Mathematical Society*, 45(1):61–75, 2008.

[56] M. Nicolau, A. Levine, and G. Carlson. Topology based data analysis identities a subgroup of breast cancers with a unique mutational profile and excellent survival. *Proc. Natl. Acad. Sci. USA*, 108(17):7265–7270, 2011.

[57] H. Edelsbrunner and J. Harer, *Computational Topology - an Introduction*. American Mathematical Society, 2010.

[58] J. Brown and T. Gedeon, "Structure of the afferent terminals in terminal ganglion of a cricket and persistent homology," *PLoS ONE*, vol. 7, p. e37278, 05 2012.

[59] H. Wagner, P. Dlotko, and M. Mrozek, *Computational Topology in Text Mining*, vol. 7309 of *Lecture Notes in Computer Science*, pp. 68–78. Springer Berlin Heidelberg, 2012.

[60] M. Mrozek, M. Żelawski, A. Gryglewski, S. Han, and A. Krajniak, "Homological methods for extraction and analysis of linear features in multidimensional images," *Pattern Recognition*, vol. 45, no. 1, pp. 285 – 298, 2012.

[61] P. Frosini and C. Landi, "Persistent betti numbers for a noise tolerant shape-based approach to image retrieval," *Pattern Recognition Letters*, vol. 34, no. 8, pp. 863 – 872, 2013. Computer Analysis of Images and Patterns.

[62] H. Edelsbrunner, *Persistent Homology in Image Processing*, vol. 7877 of *Lecture Notes in Computer Science*, pp. 182–183. Springer Berlin Heidelberg, 2013.

[63] G. Carlsson, "Topology and Data," *BULLETIN (New Series) OF THE AMERICAN MATHEMATICAL SOCIETY*, vol. 46, no. 2, pp. 255–308, 2009.

[64] J. M. Chan, G. Carlsson, and R. Rabadan, "Topology of viral evolution," *Proceedings of the National Academy of Sciences*, vol. 110, pp. 18566–18571, Nov. 2013.

[65] J. A. Perea and J. Harer, "Sliding windows and persistence: An application of topological methods to signal analysis."

[66] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer, "Stability of persistence diagrams," *Discrete Comput. Geom.*, vol. 37, pp. 103–120, Jan. 2007.

[67] "Phat (persistent homology algorithm toolbox)." `https://code.google.com/p/phat/`.

[68] "Capd::redhom.." `http://redhom.ii.uj.edu.pl/`.

[69] G. Csardi and T. Nepusz, "The igraph software package for complex network research," *InterJournal*, vol. Complex Systems, p. 1695, 2006.

[70] "Celery: Distributed task queue." `http://www.celeryproject.org/`.

[71] "Redis." `http://redis.io/`.

[72] "Rabbitmq." `http://www.rabbitmq.com/`.