Persistent Homology and Map Digitization

Sitan Chen

I. Introduction

A. Map Digitization

The field of geoinformatics is concerned with the retrieval, storage, analysis, and visualization of spatial data. In light of the rising need for highly modularized, easily navigable digital spatial data among earth scientists and geographers, coupled with the abundance of undigitized physical maps, it is becoming increasingly crucial to develop fast and robust methods for automatically converting these physical maps into efficient digital representations.

Lee and Su [9] identify three levels of map digitization stratified by complexity of the features extracted, the one highest in feature complexity being the task of segmentation and recogniton, respectively extracting graphical features like borderlines and textual features like building names. We first review some prior work related to this task.

With respect to recognition in maps, most major work has been in the domain of text detection rather than OCR and has drawn on techniques in the domain of connected component analysis, the algorithm due to Fletcher and Kasturi [4] being one of the major players in this class of methods. Tombre et al. [16] adapted Fletcher and Kasturi's algorithm to text detection in graphics-rich documents like engineering drawings and maps by building in a constraint on the possible size of a text component. Cao and Tan [1] likewise achieved separation of text with connected component analysis by noting that the lines constituting textual symbols in a map are typically shorter than the lines constituting graphical information. Li's [8] key addition to this technique was to process lines and text in a map in conjunction, using street line orientations to inform judgments about groupings of characters given by connected components.

With regards to segmentation in maps, there has been a significant body of work centered around vectorization of borderlines. Ramachandran [15] used run length encoding to compress map images into a representation that preserves the relevant borderline information. On the other hand, papers like those of Masavi et al. [12] and Lee and Su [9] process borderlines by "thinning" them to single-pixel width and directly computing geometric properties of these borderlines like endpoints and intersections. Kaneko [6] adopted the approach of identifying landmark points on contours called "line cues" and fitting lines to these cues.

With respect to segmentation *and* recognition, in [17], [18], Yamada et al. developed an algorithm which achieves both by detecting lines in maps using a directional edge detector, digitally reconstructing those strokes using morphological operations like erosion and dilation, and separating them from the remaining shapes in the map which should be text. The collection of operations in the so-called "MAP" (multi-angled parallelism) algorithm was refined in the subsequent works Pezeshk and Tutwiler [14] in order to classify sides of characters and short lines more accurately.

We will focus on the problem of segmentation and propose a solution that represents a marked departure from the above techniques. The crux of our approach will be a topological rather than a geometric one, and in the next subsection we sketch the main ideas for the body of work we will draw upon, deferring a rigorous treatment of the definitions involved to a later section.

B. Computational Topology

Topology is the study of classifying spaces up to *homeomorphism*, i.e. continuous deformation. The point of algebraic topology is to carry out this classification using *algebraic invariants*. Specifically, we want to assign to spaces some algebraic objects, for example the group of homology classes of a space, such that two spaces which are equivalent to each other up to continuous deformation get assigned the same algebraic object. To check whether two spaces are distinct, it is thus sufficient to check, for example, that their homology groups are not isomorphic.

The motivation for applying topology to computer science is that many problems in data mining, machine learning, and computer vision often involve the analysis of large point cloud data sets. In our setting, say we wanted to study some scanned image of a paper map given by a cloud of black pixels $\mathcal{P}$. If we assume the points in $\mathcal{P}$ are sampled from some underlying topological space X, e.g. a smooth manifold, then to study this space we can just study the algebraic invariants on X. Of course, we don't actually have this space readily available, merely an approximation given by $\mathcal{P}$, but out of any such $\mathcal{P}$ we can construct a filtration of *chain complexes* $\{C(\epsilon)\}_\epsilon$ parametrized by the "scale" at which $\mathcal{P}$ might approximate a space. Each of these chain complexes can be geometrically realized as a topological space $X(\epsilon)$, so to study the topological features of our scanned map, it suffices to study the algebraic invariants of any one of these $X(\epsilon)$ for a good choice of ϵ.

Unfortunately, it turns out that homeomorphism can be a fairly weak measure of equivalence between topological spaces: it's an old but relevant joke that a topologist can't tell the difference between his coffee mug and a donut [1].

The key development shortly after the turn of the century that marked the inception of computational topology as a viable approach to analyzing big data provided the answer to this concern. Edelsbrunner et al. [3] conceived the notion of *persistence*, which roughly speaking calls for one to consider 1) not just a single $X(\epsilon)$, but *every* $X(\epsilon)$, 2) not just every homology class among all the $X(\epsilon)$, but those which "persist" the longest as we go across the spectrum of possible scales ϵ.

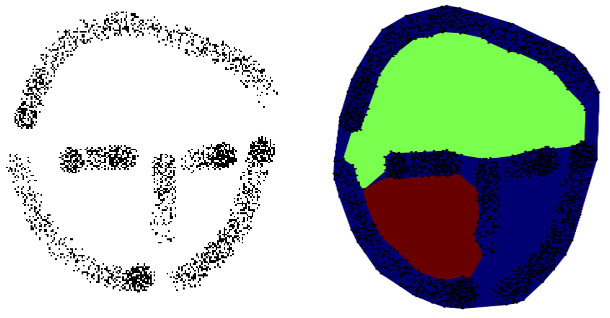
The implication is that with algebraic topology, one can visualize entire datasets with just a simple "barcode" of lines, one for each homology class i, with length equal to the range of scales at which i persists.

As the level of generality of this approach suggests, persistence and computational topology have found far-reaching applications in everything from breast cancer treatment [13] to gait analysis [7] to natural image classification [5].

The application we will focus on is to the auto-completion of contours [10]. Again, we will describe the topological algorithm for this in full rigor in subsequent sections, but for the sake of stating the primary contributions of this work, it will suffice to sketch this applications.

In [10], Kurlin developed an $O(n \log n)$ algorithm for inferring the contours from which a noisy 2D point cloud $\mathcal{P}$ of size n is sampled. The algorithm computes the persistent homology classes of the filtration of so-called *dual α-complexes* of $\mathcal{P}$, and each class turns out to represent the interior of a major contour in the image. Figure 1 one such noisy cloud $\mathcal{P}$ as well as the output of our implementation of Kurlin's algorithm.

[1]The space given by the surface of a donut and the space given by the surface of a coffee mug are homeomorphic to the torus $S^1 \times S^1$



(a) Noisy point cloud $\mathcal{P}$ (b) Output of our implementation of Kurlin's algorithm

Fig. 1: Kurlin's contour autocompletion algorithm

C. Our Contributions

While it seems unlikely at this point that such a technique would drastically outperform old, better-known approaches, especially given that map segmentation has been so widely studied, one can certainly ask whether a topological approach is at all feasible.

The primary goal of this work was thus to develop a proof-of-concept implementation of map segmentation via persistent homology. This work's main contributions are a pipeline for map segmentation making use primarily of the topological methods sketched above.

We will use Kurlin's algorithm [10] for the auto-completion of contours in a noisy point cloud to achieve segmentation and also use a simple non-topological method which we briefly mention later to facilitate this by removing extraneous text. We will work with cadastral maps of land plots [2] for simplicity so that the abovementioned text detection algorithm works without major issues. Our algorithm takes as input a scanned cadastral map, and outputs a segmentation $\mathcal{S}$ of the map into the regions depicted.

Beyond this pipeline, our other contribution is a demonstration that costly topological computations can sometimes be made more efficient by reducing them to quicker local computations and then extracting global topological information from these local pieces. As a consequence of this idea, we obtain an improvement in runtime for Kurlin's algorithm from $O(n \log n)$ to $O(n)$ under the assumption that the closed contours of the point cloud data given as input are all small.

D. Organization

In section II, we formalize the constructions sketched above and establish the theoretical background for our digitization algorithm. In section III,

[2]See http://gis.atlantaga.gov/apps/lots/ for the actual collection of cadastral maps used in this work.

we give a full description of our pipeline for map digitization. In section III, we conduct an informal analysis of the performance of our implementation.

II. Preliminaries

This section is merely to place all of our work on a solid theoretical footing, and the information presented in subsections II-A to II-C are available in any standard computational topology survey and can be skimmed without detriment to the reader's understanding of subsequent sections.

A. Complexes and Filtrations

"Combinatorics is the slums of topology."
-J.H.C. Whitehead

Simplicial complexes can be thought of as a generalization of the notion of a graph.

Definition II.1. A *simplicial complex* $K = (Z, \mathcal{F})$ consists of the data of a vertex set Z and a family S of finite sets $\sigma \subseteq Z$ closed under taking subsets, that is, if $\sigma \in S$, then for any $\tau \subseteq \sigma$, $\tau \in S$.

We call each such a σ a *k-simplex*, or a *simplex of dimension k* if $|\sigma| = k + 1$, and if the maximal k for which a simplicial complex contains a k-simplex is n, then we say that K is a simplicial complex *of dimension n*.

If σ consists of the vertices $v_0, ..., v_k$, then the *boundary* $\partial\sigma$ of the k-simplex σ is the set of all $(k-1)$-simplices σ' consisting of the vertices $\{v_i\}_{i \in S}$ for any $S \subseteq \{0, ..., k\}$ of size k.

We will call sometimes call 0-, 1-, and 2-simplices vertices, edges, and triangles respectively. Note that a simplicial complex of dimension 1 is just a graph.

Definition II.2. The *n-skeleton* of a simplicial complex $K = (Z, \mathcal{F})$ is defined to be $K^{(n)} = (Z, \mathcal{G})$ where $\mathcal{G} \subseteq \mathcal{F}$ consists of all k-simplices in $\mathcal{F}$ for $k \leq n$.

We will be working with the following examples of simplicial complexes you could associate to a point cloud $\mathcal{P} \subset \mathbb{R}^2$.

Example II.1 (Delaunay complex)**.** *The* Delaunay complex $\text{Del}(\mathcal{P})$ *of a point cloud $\mathcal{P}$ is the two-dimensional simplicial complex whose 0-simplices are the points of $\mathcal{P}$, whose 2-simplices consist of all triangles σ with vertices $u, v, w \in \mathcal{P}$ for which the circumcircle contains no other vertices of $\mathcal{P}$, and whose 1-simplices are merely all edges of triangles in* $\text{Del}(\mathcal{P})$.

Example II.2 (Dual Delaunay Complex)**.** *The* dual Delaunay complex $\text{Del}^*(\mathcal{P})$ *of a point cloud $\mathcal{P}$ is the one-dimensional complex which has a 0-simplex for each $\sigma \in \text{Del}(\mathcal{P})$, and a 1-simplex for every pair of $\sigma_i, \sigma_j \in \text{Del}(\mathcal{P})$ which share an edge in $\text{Del}(\mathcal{P})$.*

The point of introducing Del^* is that it captures essentially the same information as Del but has no 2-simplices. We will be using the dual Delaunay complex for auto-completion of contours.

B. Filtrations

Definition II.3. A *subcomplex* $L \subseteq K$ of a simplicial complex $K = (Z, \mathcal{F})$ is any simplicial complex whose set of vertices and set of simplices are subsets of Z and $\mathcal{F}$ respectively. A filtration of a complex K is a (possibly infinite) sequence of subcomplexes $K_0, ..., K_m$ such that

$$\emptyset = K_0 \subseteq K - 1 \subseteq \cdots \subseteq K_m = K.$$

We can now make rigorous our notion of approximating a space at various scales by defining scale in terms of a function from a simplicial complex to the reals.

Definition II.4. For a simplicial complex $K = (Z, \mathcal{F})$, define a *filter function* to be any map $f : K \to \mathbb{R}_{\geq 0}$. The *filtration of K given by f* is the uncountably infinite sequence $\{K(\epsilon)\} = \{(Z(\epsilon), \mathcal{F}(\epsilon)\}$ where $Z(\epsilon) = \{z \in Z \mid f(z), \epsilon\}$ and $K^{(n)}(\epsilon)$ is recursively defined to be $K^{(n-1)}(\epsilon)$ together with all n-simplices σ of $\mathcal{F}$ such that $\partial\sigma \in K^{(n-1)}(\epsilon)$ and $f(\sigma) < \epsilon$.

Example II.3. *For $K = \text{Del}(\mathcal{P})$, we could simply define the filter function f to be $f(v) = 0$ for vertices v, $f(\{u, v\}) = \|u - v\|_2$, and $f(\sigma) = \max_{\tau \subset \sigma} f(\tau)$ for σ of dimension greater than 1. Denote $\text{Del}(\epsilon)(\mathcal{P})$ by $\mathcal{P}(\epsilon)$. $\mathcal{P}(\alpha)$ is the* α-complex *of $\mathcal{P}$ and gives a filtration $\emptyset = \mathcal{P}(0) \subseteq \cdots \subseteq \mathcal{P}(\infty) = \text{Del}(\mathcal{P})$ of the Delaunay complex.*

Example II.4. $\text{Del}^*(\mathcal{P})$ *doesn't necessarily have an embedding in $\mathbb{R}^2$, but define the filter function f to be $f(v) = 0$ for vertices v and $f(\{u, v\}) = -d(u, v)$, where $d(u, v)$ denotes the length of the common (longest) side of the triangles in $\text{Del}(\mathcal{P})$ corresponding to u, v. For $\alpha > 0$, denote $\text{Del}^*(-\alpha)(\mathcal{P})$ by $\mathcal{P}^*(\alpha)$. $\mathcal{P}^*(\alpha)$ is the* dual α-complex *of $\mathcal{P}$ and gives a filtration $\emptyset = \mathcal{P}^*(\infty) \subseteq \cdots \subseteq \mathcal{P}^*(0) = \text{Del}^*(\mathcal{P})$ (note that the indexing of the filtration goes in the opposite direction because $\text{Del}^*(\mathcal{P})$ is dual to $\text{Del}(\mathcal{P})$).*

C. Persistent Homology

Now that we have filtrations of simplicial complexes, we can finally attach algebraic invariants to our point cloud $\mathcal{P}$. All we need is the notion of a "chain complex" and a "filtered chain complex." The following assumes working knowledge of group theory, but we include these technical details only for completeness. Again, the reader is welcome to merely skim over this if he/she would like, as it is not crucial to the understanding of our contributions in this work.

Definition II.5. A *chain complex* C consists of the data of a countably infinite collection of abelian groups $\{A_n\}_{n\geq 0}$ and, for each n, a map $\partial_n : A_n \to A_{n-1}$ such that $\partial_n \circ \partial_{n+1} = 0$ for all n. In other words, C is the datum of a sequence

$$\cdots \to A_n \xrightarrow{\partial_n} A_{n-1} \xrightarrow{\partial_{n-1}} \cdots \xrightarrow{\partial_2} A_1 \xrightarrow{\partial_1} A_0 \xrightarrow{\partial_0} 0$$

where the composition of any two consecutive arrows is zero.

A map of chain complexes $F : C \to C'$ is a collection of maps $\{F_n : A_n \to A'_n\}$ for which the diagram

$$\begin{tikzcd}
\cdots \arrow[r] & A_n \arrow[r, "\partial_n"] \arrow[d, "F_n"'] & \cdots \arrow[r, "\partial_1"] & A_0 \arrow[r, "\partial_0"] \arrow[d, "F_1"'] & 0 \\
\cdots \arrow[r] & A'_n \arrow[r, "\partial_n"] & \cdots \arrow[r, "\partial_1"] & A'_0 \arrow[r, "\partial_0"] & 0
\end{tikzcd}$$

commutes.

A *filtered chain complex* C consists of the data of a countably infinite collection of chain complexes $C(\epsilon)$ and, for each $\epsilon < \epsilon'$, a map of chain complexes $F^{\epsilon,\epsilon'} : C(\epsilon) \to C(\epsilon')$.

As one final step before we define persistent homology, we see how we can extract a filtered chain complex out of a point cloud $\mathcal{P}$.

Definition II.6. For a simplicial complex K, define K_n to be the free abelian group generated by the set of n-simplices in K, i.e. the collection of all finite formal sums $\sum \sigma_i$ of some n-simplices σ_i endowed with the natural group structure.

For each $n \geq 0$, define $\partial_n : K_n \to K_{n-1}$ sending any n-simplex $\sigma = \langle v_0, ..., v_n\rangle$ to $\sum_{i=0}^{n}(-1)^i\langle v_0, ..., \hat{v}_i, ..., v_n\rangle$, where $\hat{v}_i$ denotes omission of v_i and where ∂_n is defined for the rest of K_n by extending linearly. We get the *chain complex associated to* K:

$$\cdots \to K_n \xrightarrow{\partial_n} K_{n-1} \xrightarrow{\partial_{n-1}} \cdots \xrightarrow{\partial_2} K_1 \xrightarrow{\partial_1} K_0 \xrightarrow{\partial_0} 0.$$

Now if we had a filtration $\{K(\epsilon)\}$ of K given by filter function f, we have natural inclusion maps $i_n^{\epsilon,\epsilon'} : K_n(\epsilon) \hookrightarrow K_n(\epsilon')$ for all n and $\epsilon' \geq \epsilon$, and it is straightforward to check that all the $i^{\epsilon,\epsilon'} = \{i_n^{\epsilon,\epsilon'}\}$ are in fact maps of chain complexes. We thus get the *filtered chain complex given by* f.

We can now define homology and persistent homology. The former should be thought of as "holes" in $\mathcal{P}$ at some scale, while the latter should be thought of as holes in the actual space X that $\mathcal{P}$ approximates.

Definition II.7. [Homology] The *homology groups* $\{H_*(C)\}$ of a chain complex C are given by $H_n(C) = \mathrm{Ker}(\partial_n)/\mathrm{Im}(\partial_{n+1})$ (note that the fact that $\partial_n \circ \partial_{n+1} = 0$ guarantees that $\mathrm{Im}(\partial_{n+1}) \subseteq \mathrm{Ker}(\partial_n)$ so that this quotient actually makes sense). The elements of the homology groups of C are called *homology classes*.

Remark II.1. *It is a simple exercise in homological algebra to check that a map of chain complexes $F : C \to C'$ induces maps on homology $\{\tilde{F}_n : H_n(C) \to H_n(C')\}$.*

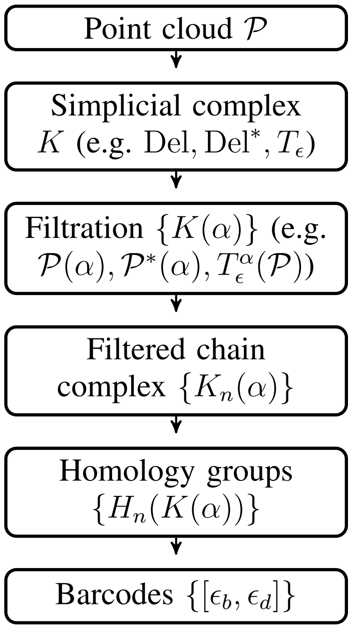
Definition II.8. [Persistent homology] Given a filtered chain complex $\{C(\epsilon)\}$, a homology class $\gamma \in H_n(C(\epsilon_b))$ is said to be *born at scale* ϵ_b if for all $\epsilon < \epsilon_b$, γ is not in the image of any of the maps $H_n(C(\epsilon)) \to H_n(C(\epsilon_b))$.

γ is said to *die at scale* ϵ_d if for some $\epsilon < \epsilon_b$, the image of γ under $H_n(C(\epsilon_b)) \to H_n(C(\epsilon_d))$ merges into the image of $H_n(C(\epsilon)) \to H_n(C(\epsilon_d))$.

The *persistence* of γ is defined to be $\epsilon_d - \epsilon_b$.

The collection of intervals $[\epsilon_b, \epsilon_d]$ for all nth homology classes $\gamma \in H_n(C(\epsilon))$ for some ϵ form the *nth bar code of* $\mathcal{P}$.

The upshot is that out of a point cloud $\mathbb{P}$ and a filter function f, if we chase along the sequence of steps depicted in Figure 2, we get a collection of intervals that can be used to visualize the most important topological information about our point cloud and the space that it approximates.



Point cloud $\mathcal{P}$
Simplicial complex K (e.g. Del, Del*, T_ϵ)
Filtration $\{K(\alpha)\}$ (e.g. $\mathcal{P}(\alpha), \mathcal{P}^*(\alpha), T_\epsilon^\alpha(\mathcal{P})$)
Filtered chain complex $\{K_n(\alpha)\}$
Homology groups $\{H_n(K(\alpha))\}$
Barcodes $\{[\epsilon_b, \epsilon_d]\}$

Fig. 2: Steps for obtaining barcodes

III. Methods

Having laid the theoretical basis for this work, we can finally describe our pipeline for map digitization consisting of the following steps: 1) split the input into small overlapping $W \times W$ windows $W_{i,j}$, and for each window: 2) detect and remove text pixels, 3) compute a segmentation of the window into disjoint regions bounded by borderlines, 4) paste together patches.

A. *Windows*

The motivation for running our algorithm locally on individual windows rather than globally on the entire map is that we want to take advantage of the fact that in theory, any significant topological features of the scanned map are restricted in size to whatever the largest bounded region on the map is.

In fact this gives us improvement by a logarithmic factor on expected running time: whereas a cloud of n points would require $O(n \log n)$ to output a segmentation under a global application of our algorithm, computing $O(n)$ local segmentations on constant-sized windows takes $O(n)$ time, even including the additional processing that we describe in subsection III-D to reconstruct a global segmentation out of local ones.

Concretely, we cover the scanned image with a grid of windows $\{W_{i,j}\}$ overlapping by a factor of $\lambda = 1/4$ (two windows W and W' *overlap by a factor of* λ if $\frac{|W \backslash W'|}{|W|} = \lambda$), where the top-left corner of Wi, j is the pixel in row $W\lambda i$ and column $W\lambda j$. Each of these windows comes equipped with a natural cloud $\mathcal{P}_{i,j}$ of points corresponding to pixels of sufficiently high intensity.

B. *Text Removal*

To detect text, we take the approach of Maguluri et. al in [11] of training a support vector machine classifier based on the histogram of oriented gradients (HoGs) around each point. Their motivation for using HoGs as features was that the support of a HoG around a non-textual pixel was typically much smaller than one around a textual pixel. We outline the details for our implementation of this approach, though we should stress that this step is not the focus of this work, merely a necessary tool.

First, let's recall relevant definitions. First, we want to compute the magnitude of the gradients for our image, so convolve our image with small derivative of Gaussian kernels in both horizontal and vertical directions and use the resulting x- and y-gradients to compute a gradient matrix giving the magnitude of the gradient at each point as well as an orientation matrix giving the direction of the gradient vector at each point. Fix an odd width n and consider the $n \times n$ neighborhood N centered at a particular pixel of our image. The d bins of our histogram are indexed by ranges of angles $[\theta_0, \theta_1), ..., [\theta_{d-2}, \theta_{d-1}), [\theta_{d-1}, 2\pi)$, where $\theta_i = 2\pi i/d$ for $i = 0, ..., d-1$. The height of the ith bin is the number of pixels in N lying within the appropriate range $[\theta_i, \theta_{i+1})$.

To avoid the possibility of pixels in what appear to be homogeneous image regions having nontrivial gradient magnitude, we follow Magluri et. al in thresholding our gradient map by some appropriately chosen value k.

Using online data provided by Maguluri et al. at http://www.public.asu.edu/~bli24/icassp2013.html in the form of labeled images of floor plans, we trained an SVM for classifying pixels in an image as belonging to text or otherwise. As observed in [11] however, the decisions of the SVM alone lead to fuzzy detected regions which include many extraneous non-text pixels, so our text detection algorithm only counts as text those pixels which are both positively classified by our SVM and have gradient magnitude higher than our threshold k. As one more precaution against false positives, we also add the condition that the HoG around the pixel in question has a sufficiently large support.

C. *Segmentation*

At this stage of the algorithm, essentially all of the text in the image has been removed, but because some non-text pixels were still erroneously classified as text and removed in the previous, what remains in each window is a somewhat sparse sampling of the window we started with.

To split this sampling into the regions delineated by borderlines in our map, we first smooth the patch of pixel intensities with a Gaussian kernel of standard deviation $\sigma = 0.6$, collect all points $\mathcal{P}$ of sufficiently high intensity, add ϵ-noise for ϵ uniformly randomly sampled from $[0, 1]$ to both coordinates of each point in $\mathcal{P}$, and apply Kurlin's contour auto-completion algorithm from [10] to fill in the sparse borderlines in $\mathcal{P}$.

Roughly speaking, Kurlin's algorithm works as follows:

1) Compute the filtered chain complex given by the dual Delaunay complex
2) Form groups out of vertices of the Delaunay complex corresponding to adjacent triangles which are alive at the same time
3) Identify the most persistent groups and merge all other groups into these.

To this end, define the following two data structures: FOREST and BARCODE.

The elements v of FOREST correspond to vertices of $\mathrm{Del}^*(\mathcal{P})$ and thus to triangles T_v in $\mathrm{Del}(\mathcal{P})$. Because we would like to establish relations among these v that capture the groups formed in step 2) as well as the order in which they were formed. To keep track of this order, we will model FOREST as union-find data structure. We endow each element v with the following attributes:

- $\mathrm{birth}(v)$ is the largest scale α at which the corresponding triangle σ in $\mathrm{Del}(\mathcal{P})$ is a triangle in $\mathcal{P}(\alpha)$
- $\mathrm{parent}(v)$ is a pointer to the parent of v in FOREST
- $\mathrm{height}(v)$ is the height of the tree rooted at v
- $\mathrm{live}(v)$ is the number of its descendants (including itself) that are alive
- $\mathrm{bar}(v)$ is the index of the element in BARCODE which contains T_v (see below).

On the other hand, BARCODE is a list of elements c each representing a connected component in $\mathcal{P}^*(\alpha)$ for some range of α, i.e. a tree in FOREST. These are the "groups" constructed in step 2) above, and we endow each c with the following attributes:

- $\mathrm{ind}(c)$ is the index of c in BARCODE
- $\mathrm{birth}(c)$ is the scale at which c gets its first member in FOREST
- $\mathrm{death}(c)$ is the scale at which c is merged into another component
- $\mathrm{core}(c)$ is the set of all $v \in$FOREST which resided in c until c's death
- $\mathrm{heir}(c)$ is a pointer to the root of the tree corresponding to the component that absorbed c
- $\mathrm{supr}(c)$ is the index in BARCODE of the component that absorbed c

The algorithm proceeds as follows.

Initial steps: Given a cloud $\mathcal{P}$, first compute its Delaunay complex. Denote the triangles in the Delaunay complex by $\sigma_1, ..., \sigma_m$, and denote the corresponding vertices of FOREST by $v_1, ..., v_m$. As a technicality, also include an extra vertex FOREST corresponding to the external region of the point cloud.

By definition of the Delaunay complex, $v_i \in$FOREST is born at scale α equal to the circumradius of σ_i if σ_i is acute, and otherwise simply set $\mathrm{birth}(v_i) = 0$ for $i \neq 0$ and $\mathrm{birth}(v_0) = \infty$. As there are initially no components in MAP and no live triangles, initialize $\mathrm{bar}(v)$ to zero and $\mathrm{live}(v)$ to be empty for all v.

Now sort the edges of $\mathrm{Del}(\mathcal{P})$ in decreasing order of length. The algorithm simulates descending from scale ∞ to scale 0 by successively drawing edges e and taking α to be $\mathrm{length}(e)/2$.

Main loop: If FOREST is still disconnected, draw the next longest from the sorted collection of edges and set α as described above.

Denote by u, v the elements of FOREST corresponding to triangles sharing this edge, and connect u and v by an edge. We have three possibilities: 1) u and v were already alive but part of the same connected component in $\mathcal{P}^*(\alpha)$, 2) one of u or v had not been born yet, 3) u and v were already alive but part of different connected components.

In case 1, we are not changing anything about the grouping of elements of FOREST as u and v were already connected, so nothing happens.

Now assume without loss of generality that the tree containing u is younger than the one containing v, that is, the former first appears at a smaller scale than does the latter so that $\mathrm{birth}(\text{root}(u)) \leq \mathrm{birth}(\text{root}(v))$.

In case 2, this means that u is the node that hasn't been born yet, implying that u corresponds to a non-acute triangle. Add u to the tree in FOREST containing v, setting $\mathrm{parent}(u)$ to be the root r of this tree, $\mathrm{birth}(u) = \mathrm{birth}(r)$. If v has already died, then u joins this dead component so that $\mathrm{bar}(u) = \mathrm{bar}(v)$. Otherwise, u is appended to the list of live descendants of r.

In case 3, because we are merging two trees in FOREST, the younger component dies when it gets absorbed the older, more persistent component. Whenever we are in case 3, we thus have to add a new element c to MAP. We set $\mathrm{birth}(c) = \mathrm{birth}(\text{root}(u))$ and $\mathrm{death}(c) = \alpha$, $\mathrm{core}(c) = \mathrm{live}(\text{root}(u))$, kill each node w in the tree containing u, and set each $\mathrm{bar}(w)$ to be the index in MAP of this new component. For now, we will also set $\mathrm{heir}(c)$ to point to the first live node in the tree containing v so that by the end of the loop, we will be able to access the index of the component in MAP that absorbed c by finding $\mathrm{bar}(\mathrm{heir}(c))$.

Finally, after inserting this c into MAP, we link together our two trees, setting the parent of the root of the shorter tree to be the root of the taller one and updating heights and $\mathrm{live}(v)$ accordingly if necessary.

After this loop terminates, there is one more component to add to MAP, namely the one containing v_0. We go through the same procedure for adding a component c to MAP as above but do not set an heir as this is the last surviving component.

Segmentation: The initial segmentation is given merely by grouping all triangles in $\mathrm{core}(c)$ together for each individual c. Sort the elements of MAP in decreasing order of persistence $\mathrm{pers}(c) := \mathrm{birth}(c) - \mathrm{death}(c)$

and Find the index of the biggest drop in persistence $\mathrm{pers}(c_i) - \mathrm{pers}(c_{i+1})$ between adjacent elements $c_i, c_{i+1} \in$MAP. Our final segmentation will consist of $m := i + 1$ regions, including the external region. For each component c after the first m in MAP, starting from the last element of MAP, determine the index of the component c' that absorbed c and move $\mathrm{core}(c)$ into $\mathrm{core}(c')$. At the end, all cores will have been moved to the first m entries of MAP, and each $\mathrm{core}(c)$ forms a region in our segmentation.

Remark III.1. *This algorithm can be carried out in $O(n \log n)$ time, the bottleneck being the computation of the Delaunay complex and sorting of the triangles at the beginning.*

The output of this algorithm is a collection of "segmentation matrices" $M_{i,j}$, one for each window $W_{i,j}$ of the scanned map, where the entries of $M_{i,j}$ take on values in $\{0, ...m_{i,j}\}$, where $m_{i,j}$ is the number of regions in the segmentation of $W_{i,j}$. The (k, ℓ)th entry of $M_{i,j}$ is 0 if the corresponding pixel lands in none of the triangles of the Delaunay complex of $\mathcal{P}_{i,j}$, and otherwise equal to $1 \leq m \leq m_{i,j}$ if the corresponding pixel belongs to the mth segment in $W_{i,j}$.

D. Reconstruction

The last step is to take the output of our segmentation algorithm on each overlapping patch and merge these segmentations into a coherent segmentation of the entire map.

This step takes as input a collection of "segmentation" matrices as described above and attempts to consolidate topological features that overlapping windows share into a single segmentation matrix over the entire scanned image.

To motivate our procedure for accomplishing this, we first explain why we need to consider overlapping windows rather than disjoint ones. One issue with computing local segmentations is that some contours in a window, especially those near the edges, are not closed because they continue outside of the window. To our segmentation algorithm, such contours might as well not exist, and as a result in many windows we obtain segmentations like that of Figure **??**.

On the one hand, this obviously prevents us from reconstructing a global segmentation out of local segmentations of overly small windows. On the other, this also implies that most of the meaningful topological information in a window is concentrated away from the edges. If we were to work with disjoint windows, our local segmentation would provide very little topological data about points on the boundaries of the windows. With large enough overlaps and large enough windows however, each point is positioned close to the center of *some* window, making local-to-global reconstruction possible.

Now we make concrete the notion of "consolidating" overlapping segmentations. Concretely, say we have two adjacent, overlapping windows W, W' with segmentations respectively containing m and m' persistent regions. The key task is to determine which, if any, pairs of regions correspond. Define the matrix $I_{m \times m'}$ such that $I_{i,i'}$ is the number of pixels in $W \cap W'$ which belong to both region i in W and region i' in W'.

Then to determine whether regions i and i' correspond, we simply check whether

$$\frac{I_{i,i'}}{\sum_{j'=1}^{m'} I_{i,j'}} \geq \lambda, \quad \frac{I_{i,i'}}{\sum_{j=1}^{m} I_{j,i'}} \geq \lambda$$

for some appropriately chosen λ, that is to say, the fraction of pixels in region i that are in regions of W' other than i' is extremely small, and vice versa.

With this in mind, we can reconstruct a global segmentation by starting with a local segmentation at the top, leftmost window and building up the global segmentation by propagating down and to the right, each window contributing a small amount.

Formally, if λ is the factor of overlapping, S denotes our final segmentation matrix, and w, h denote the width and height of the map, write $a = W(1 - \lambda)/2$ and $b = W(1 + \lambda)/2$. Our reconstruction is detailed in 1:

IV. Results

Unfortunately, as the collection of maps available at http://gis.atlantaga.gov/apps/lots/ is not hand-annotated with features like total number of regions, there is no efficient way to evaluate how a standard map segmentation algorithm compares to this work's relative to ground truth. That each of these maps is also rather large (on the other of 7000×6000 pixels) makes this sort of comparison even more unrealistic.

Instead, we will hone in on a couple of sample inputs and explain some of the qualitative features and shortcomings of our segmentation algorithm.

Consider the input in Figure 3. This was broken down into a 5×5 grid of windows overlapping with factor 1/4. While our segmentation algorithm did a good job for the most part in classifying the regions on the left half of the page, two issues immediately stand out: 1) the roads are completely wrong and the regions around the road are not as consistently well detected,

Algorithm 1 Local-to-global reconstruction

```
for $i = 1$ to $w/(W\lambda)$ do
    for $j = 1$ to $h/(W\lambda)$ do
        Say regions $r_1, ..., r_k$ in $W_{i,j}$ are associated with regions $r_1^\ell, ...r_j^\ell$ in $W_{i-1,j}$ and $r_{j+1}^u, r_k^u$ in $W_{i,j-1}$
        $M_{i,j}[p_1, p_2] := r_i^d$ for each $i$ and $(p_1, p_2) \in r_i$ and appropriate $d \in \{u, v\}$.
        if $1 < i < w/(W\lambda)$ then
            colrange$(i, j) := (i + a : i + b)$
        else if $i = 1$ then
            colrange$(i, j) := (1 : b)$
        else
            colrange$(i, j) := (i - b + 1, i)$
        end if
        if $1 < j < h/(W\lambda)$ then
            rowrange$(i, j) := (j + a : j + b)$
        else if $j = 1$ then
            rowrange$(i, j) := (1 : b)$
        else
            rowrange$(i, j) := (j - b + 1, j)$
        end if
        Set chunk $C_{i,j}$ of $S$ to be
            $M_{i,j}[\text{colrange}(i, j), \text{rowrange}(i, j)]$
    end for
end for
```

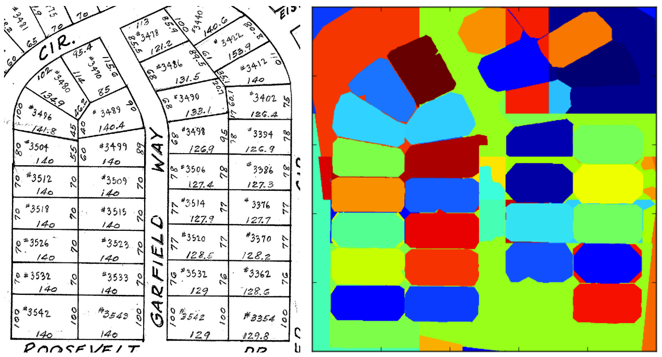


Fig. 3: Input to and output of segmentation algorithm

2) the third column of regions, particularly around the top-right corner was segmented quite poorly.

Zooming in on the window $W_{2,2}$ which has the road "Garfield Way" running through its center, we see in Figure 4 that our text detection algorithm failed to completely remove the road name. As a result, in the local segmentation, the gap between "Way" and "Garfield" was interpreted as a region in its own right (shown in red). As a remark, the segmentation in $W_{2,2}$ otherwise went according to plan more or less, as none of the blue regions are bounded by a fully closed contour, and the only regions that are bounded by a closed contour were indeed detected as persistent regions.

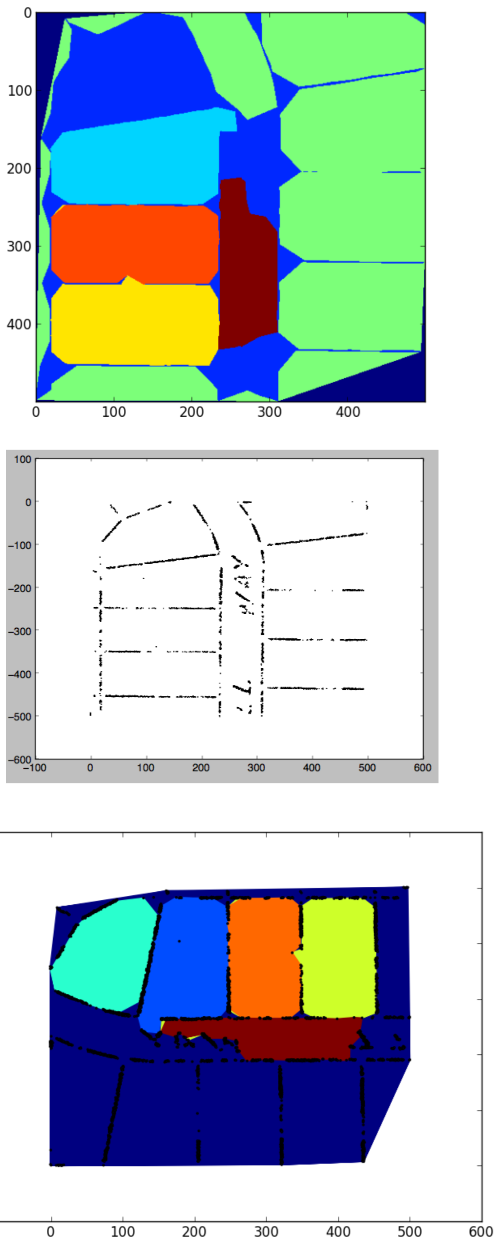


Fig. 4: Explanation of artifacts found on "Garfield Way"

We run into identical issues with "Roosevelt Drive" on the bottom of the map and with the disappearnce of plot 3532 from our segmentation.

On the other hand, the regions in the top-left corner of the map in Figure 3 are not bounded by closed contours explaining why they were absorbed into the red region in the top-left of our segmentation.

As for the faulty segmentation in the top right-hand corner, we observe what seems to be a failure on the part of our reconstruction algorithm to patch together persistent regions in different windows. The reason for this failure is simply that plot number 3440 in the top-right corner is also not bounded a closed contour and thus doesn't actually belong to any persistent region.

V. Conclusion

While the accuracy of our segmentation algorithm in detecting consistently shaped, relatively noise-free regions like the rectangular regions in the cadastral maps we used is promising, it is a bit too sensitive to noise, e.g. unremoved text pixels, and as a result we have to compensate by lowering the criteria for the kinds of pixels we filter out.

A relatively straightforward direction for immediate future work would be simply to lessen the noise that our text remover leaves behind. After all, this is easily the part of our pipeline with the greatest room for improvement. One of the issues with our detection algorithm was that the parameters were selected manually without conducting any exhaustive searches over the parameter space, something we hope to remedy in future iterations. Another issue was that computing HoG descriptors for every pixel of an image was a bit too costly both in space and time given the sheer size of the maps we were dealing with.

As we chose the algorithm in [11] for text detection primarily for its simplicity, however, it might be more productive to consider other potentially more powerful approaches like the ones mentioned in the introduction, e.g. the algorithm of Fletcher and Kasturi [4], connected component analysis processing lines and text simultaneously, etc.

With respect to employing other topological methods, one of the original intentions of this project was to add another layer to this pipeline for achieving character recognition using techniques introduced by Collins et al. in [2], drawing upon ideas in both algebraic topology and differential geometry to construct a descriptor for curved shapes. The complex they associate to a point cloud $\mathcal{P}$ is the *tangent complex* consisting of the fibers of each point in the bundle of tangent spaces over $\mathcal{P}$, and the "scale" ϵ that they filter by is the curvature of each point $x \in \mathcal{P}$. With this descriptor, they are able to implement a character recognition algorithm for classifying a small subset of the letters of the alphabet. As the regions in the cadastral maps we are interested in are annotated by numerical data, a natural next step would be to implement a digit classifier using Collins et al.'s approach. Some of this code has already been written, but the code for computing the persistent homology of the filtered tangent complex using the algorithm of [19] is incomplete.

All of the code for this project can be found at https://github.com/sitanc/digimap.

References

[1] R. Cao and C.L. Tan. Text/graphics separation in maps. *Proceedings of 4th IAPR International Workshop on Graphics Recognition, Kingston*, Ontario(Canada), pp. 44-48, September 2001.

[2] A. Collins, A. Zomorodian, G. Carlsson, and L. Guibas, A barcode shape descriptor for curve point cloud data, C*omputers and Graphics*, Volume 28, 2004, pp. 881894.

[3] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete Comput. Geom.*, 28:511533, 2002.

[4] L. A. Fletcher and R. Kasturi. A Robust Algorithm for Text String Separation from Mixed Text/Graphics Images. *IEEE Transactions on PAMI*, 10(6):910918, 1988

[5] R Ghrist. Barcodes: The persistent topology of data. *Bull. Amer. Math. Soc.*, October 2007.

[6] Toru Kaneko. Line structure extraction from line-drawing images. *Pattern Recognition* 25(9): 963-973 (1992)

[7] Javier Lamar-Leon, Edel B. Garcia Reyes, and Rocio Gonzalez-Diaz. Human gait identification using persistent homology. In Luis Alvarez, Marta Mejail, Luis Gomez, and Julio C. Jacobo, editors, CIARP, volume 7441 of Lecture Notes in Computer Science, pages 244251. Springer, 2012.

[8] L. Li, G. Nagy, A. Samal, S. C. Seth, and Y. Xu. Integrated text and line-art extraction from a topographic map. *International Journal of Document Analysis and Recognition*, 2(4):177185, 2000.

[9] Lee, L.-H. and Su, T.-T., 1996, Vision-based image processing of digitized cadastral maps. *Photogrammetric Engineering and Remote Sensing*, 62, 5, 533-538.

[10] Kurlin, V. Auto-completion of contours in sketches, maps and sparse 2D Images based on topological persistence. *Proceedings of CTIC 2014: Computational Topology in Image Context*, to appear.

[11] Hima Bindu Maguluri, Qiongjie Tian, Baoxin Li: Detecting text in floor maps using Histogram of Oriented Gradients. *ICASSP 2013*: 1932-1936

[12] Musavi, M.T., M.V. Shirvaikar, E. Ramanathan, and A.R. Nekonei, 1988. A vision based method to automate map processing, *Pattern Recognition*, 21(4):319-326.

[13] Nicolau, M., Levine, A. J. & Carlsson, G. Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. *Proc Natl Acad Sci USA* **108**, 726570 (2011).

[14] Pezeshk, A. & Tutwiler, R. L. (2010b). Improved Multi Angled Parallelism for separation of text from intersecting linear features in scanned topographic maps, *ICASSP 2010*, pp. 10781081.

[15] Ramachandran, K., 1980. Coding method for vector representation of engineering drawings, Proceeding of the IEEE, 68(7):813-817.

[16] K. Tombre et al., Text/Graphics Separation Revisited, Proc. 5th IAPR Intl Workshop Document Analysis Systems, Springer, 2002, pp. 200211.

[17] H. Yamada, K. Yamamoto, T. Saito, and S. Matsui. MAP: Multi-angled parallelism for feature extraction from topographical maps. *Pattern Recognition*, 24(6):479-488, 1991.

[18] H. Yamada, K. Yamamoto, K. Hosokawa. Directional mathematical morphology and reformalized Hough transformation for the analysis of topgraphic maps. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(4):380-387, 1993.

[19] A. Zomorodian and G. Carlsson, Computing persistent homology, *Discrete and Computational Geometry*, 33 (2), 2005, pp. 247-274. MR2121296 (2005j:55004)