Topology-Preserving Downsampling of Binary Images

Chia-Chia Chen[1,2] and Chi-Han Peng[1,3]

[1] College of AI, National Yang Ming Chiao Tung University, Taiwan
[2] s1073736@gmail.com
[3] pengchihan@nycu.edu.tw

Abstract. We present a novel discrete optimization-based approach to generate downsampled versions of binary images that are guaranteed to have the same topology as the original, measured by the zeroth and first Betti numbers of the black regions, while having good similarity to the original image as measured by IoU and Dice scores. To our best knowledge, all existing binary image downsampling methods don't have such topology-preserving guarantees. We also implemented a baseline morphological operation (dilation)-based approach that always generates topologically correct results. However, we found the similarity scores to be much worse. We demonstrate several applications of our approach. First, generating smaller versions of medical image segmentation masks for easier human inspection. Second, improving the efficiency of binary image operations, including persistent homology computation and shortest path computation, by substituting the original images with smaller ones. In particular, the latter is a novel application that is made feasible only by the full topology-preservation guarantee of our method.

Keywords: Binary Image Downsampling · Discrete Optimization · Image Segmentation

1 Introduction

Binary images, i.e., images consisting of only "foreground" (e.g., black) and "background" (e.g., white) pixels, are widely used in computer visions and computer graphics. Example usages include data structures for segmentation masks, image inputs to document analysis and industry machine vision systems [19], and the encoding of 2D geographical maps in games and animations. A classic book on machine vision [11] pointed to several advantages of binary images. First, designers noted that binary images are easier for humans to recognize key features such as silhouettes. Second, binary images led to significantly smaller memory and processing requirements of computer vision algorithms than their grey-level or color image counterparts. In particular, the binary nature of pixel values enables efficient run-length encoding of the data storage and the applicability of binary logical operations instead of integer arithmetic operations.

A key characteristic of binary images compared to grey-level or color images is their natural correspondence to 2D graphs. To do so, we first convert a binary

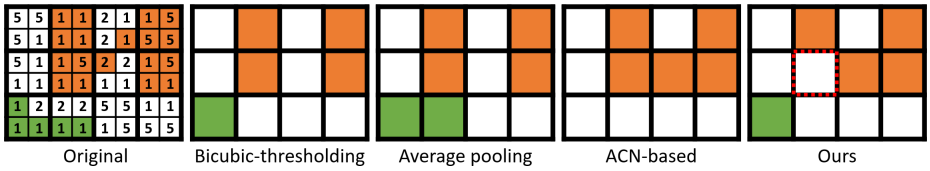

Fig. 1: Downsampling a binary image of two connected components by a factor of 2 (i.e., turning each 2x2 block of pixels into a downsampled pixel). Bicubic filtering-and-thresholding (by OpenCV), average pooling (when tie choose foreground color), and the adaptive crossing number (ACN)-based method [6], all generate results with different topology. We show the ACNs in each original pixel. Our method finds a topologically correct result. A non-trivial pixel-color decision is marked in red.

image to a regular quad mesh by taking every pixel as a quad face. We then remove all quads corresponding to white pixels and the resulting dangling vertices and edges (i.e., vertices and edges without adjacent faces). We now have a graph consisting of black-pixel quads only. This graph has zero or more connected components and zero or more holes inside each components. We denote the number of connected components and the total number of holes, namely the zeroth and first Betti numbers, as the *topology* of the binary image.

The topology of binary images plays a key role in their applications. There exists a large body of research concerning the efficient computation of topological operations on binary images such as border following, region adjacency graph, medial axis / skeletonization calculation, morphological operations (such as thinning and dilation), and distance transforms [2, 3, 11, 19, 23, 24, 26, 27]. For neural image and volume segmentation methods, it has been shown that incorporating prior knowledge about the topology (in terms of Betti numbers) of the segmented objects, usually in forms of new loss functions, can improve the segmentation results' topological accuracy and in some cases even per-pixel accuracy [5, 9, 10]. In all these applications, having topologically correct versions of a binary image at different resolutions can be useful.

However, preserving the topology when downsampling a binary image is a non-trivial and sometimes impossible task. To our best knowledge, all existing binary image downsampling methods, including bilinear or bicubic filtering-and-thresholding, various pooling-based approaches, and an adaptive crossing number-based method proposed in 2007 [6], can cause topological changes. We show examples in Figure 1. There, we can see that the color of a downsampled pixel can not be decided by simply checking the ratios of black or white pixels in its corresponding block of original pixels - *sometimes a downsampled pixel has to be white even if its corresponding block of pixels are all black.* In Figure 2 (a), we show a binary image without any topological correct downsample solutions. Note that our approach finds a non-trivial and correct solution to Figure 1 and definitely declares Figure 2 (a) to be an infeasible problem.

One can instead design a downsampling algorithm that guarantees to output topologically correct results by leveraging morphological operations. Although

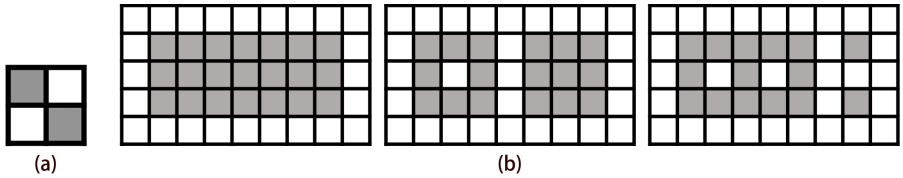

Fig. 2: (a) A binary image without any topologically correct downsampling solutions. (b) Three black regions with different Betti numbers (from left to right: (1,0), (2,1), (3,2)). However, their Euler characteristics all equal to 1. This can be verified by their numbers of vertices, edges, and faces: (32,52,21), (32,48,17), and (32,46,15). This is expected because Euler characteristic equals the number of connected components minus the number of holes for 2D graphs ([11]).

we did not find such algorithms published in research venues, we envision a baseline approach as follows. For each connected component in the original binary image, we create a corresponding trivial shape with the same topology in the downsampled image. We then iteratively improve the shapes of each component by dilation operations. We found that such a greedy approach can not produce results that have high similarity to the original binary image.

In short, the downsampling task has two main goals - the result binary image shall have the same topology and also high similarity to the original. Our insight is to formulate the task as an *discrete optimization* problem - in which the black-or-white decisions of each downsampled pixels are encoded as Boolean variables, the topology preservation is a hard constraint, and the similarity to the original binary image is the objective function to maximize. In this way, any computed solution to the optimization problem is akin to a topologically correct downsampled binary image while having good similarity to the original. Another benefit of such a optimization-based approach is that impossible tasks are identified as infeasible problems.

The bottleneck of the optimization problem design is the formulation of the topology-preserving hard constraint. As has been shown in Figure 1, topology-preservation is a global phenomena where the decision of one downsampled pixel can be influenced by regions far away. Therefore, any local approaches (i.e., making the decision based on the values of nearby pixels) would not work. We also found that constraining the Euler characteristics of each components does not work because two graphs with different Betti numbers can have the same Euler characteristic (shown in Figure 2 (b)). Inspired by the topological "no-island" constraint for network design problems in [17], we propose a novel formulation to ensure that *the topology of each component's boundary to remain a circle*, which subsequently ensures that the topology of each component to remain the same.

In summary, our contributions are as follows:

- We propose a novel discrete optimization-based approach to tackle the previously unsolved topology-preserving binary image downsampling problem. Our method outputs results that are not only topologically correct, but

also have good similarity scores that are competitive with conventional approaches without topology-preservation guarantees.
- We have designed our problem formulation such that it can be solved efficiently. In fact, our method can be used to speed up binary image operations such as persistent homology computation.
- We show that the main advantage of our method, namely full topology-preservation, enables novel applications of binary image downsampling that traditional downsampling methods could not support.

2 Related Work

2.1 Existing Binary Image Downsampling Methods

We base our discussions of existing common binary image downsampling methods on the implementations in OpenCV [4]. To find the color of a downsampled pixel at coordinate (X, Y), the **Nearest Neighbor** approach simply converts (X, Y) back to the original resolution, $(A * X, B * Y)$, A and B are the downsampling factors in width and height, and use the pixel color at $(A * X, B * Y)$. **Average Pooling** (in OpenCV it is called "**Area**") assigns black to a downsampled pixel if the ratio of black pixels in its A-by-B block equal or greater than half, and assigns white otherwise. Similarly, **Max or Min Pooling** choose a black big-pixel if any or all of the corresponding pixels are black, respectively. **Bilinear or Bicubic Interpolation-And-Thresholding** does a grey-image downsampling using the standard bilinear or bicubic interpolation first, and then convert the grey-level result to a binary image using thresholding at half grey.

In [18] and [15], the necessary and sufficient conditions for a binary image to retain topology after arbitrary rigid transformations (translation and rotation, but no scaling) are analyzed. In [16], an algorithm to do arbitrary topology-invariant affine transformations (including scaling) for binary images is proposed. However, the algorithm often could not reach a feasible solution (if one exists) for downsampling tasks. A detailed discussion is in the Supplementary Materials.

The adaptive crossing number (ACN)-based method [6] also aimed for binary image downsampling with topology-preservation as a goal. However, the method does not guarantee topologically correct results. The method only performs downsampling by a factor of 2. In short, ACN are integer numbers assigned to each pixels in the original image such that higher values indicate pixels that are more likely to alter the topology when not chosen, and vice versa. For each downsampled pixel, the color of the pixel with the highest ACN is chosen. In tie, the pixel in the first scanning order (the upper-left corner) is chosen.

2.2 Leveraging Topology for Neural Image Segmentation Methods

There exist many methods that leverage the topological properties of segmentation masks to improve the results of neural segmentation methods. They mostly do so by introducing novel loss functions that are more sensitive to the topological differences between prediction results and the ground truth. In [14], the

authors found that the feature maps of a pretrained VGG19 neural network [22] tend to be more topologically correct and designed a topological loss function using the outputs of these feature maps. In [10], a more direct and effective topological loss function is introduced that measures the differences in terms of the persistent homology (PH) of segmentation masks. Note that this method encodes only the first Betti number (numbers of connected components) of 2D segmentation masks. In [5], a more general PH-based cost function design is introduced that can encode arbitrary lengths of Betti numbers of segmented objects (e.g., numbers of cavities in volumetric data). In [9], a non PH-based cost function design, based on homotopy warping, is introduced that showed improved accuracy and computational cost. Other non PH-based topological cost function designs include one based on morphological skeleta [20] and one based on Euler characteristics [12]. Our method can benefit these methods by providing easier data visualization. Also, we envision that being able to quickly generate segmentation masks at reduced resolutions but still with the same topology (which was not possible prior to our method) can lead to novel algorithm designs.

Persistent Homology (PH). In short, PH encodes the "birth" and "death" times of each components during a consecutive session of dilation from each component being a single point until all components are so inflated to merge into a single component without holes. Unlike Betti numbers which are discrete, PH are continuous-valued and differentiable encoding. Therefore, they can be easily modeled as cost functions of neural networks. However, the main downside is that PH computation is expensive ([9, 12]). In Section 5.2, we demonstrate that our method can be used to significantly reduce the cost of PH computation with a small impact to its accuracy.

3 Our Method

We begin with definitions. A binary image consists of pixels of exactly two possible colors - foreground ("black" in short) and background ("white" in short). Note that in our figures we may use different colors to denote pixels of different connected components (e.g., Figure 1). We downsample a binary image of width W and height H by a factor of A in width and B in height. We only consider the case of A and B being positive integers and W is dividable by A and H is dividable by B. In other words, we practically convert each A-by-B block of pixels of possibly different colors into one downsampled "big"-pixel of a single color. The resolution of the downsampled binary image is W/A-by-H/B.

A W-by-H binary image can be turned into a regular grid-like quad mesh of H rows and W columns of quads. A quad is black or white if it corresponds to a black or white pixel, respectively. If we remove all the white quads and the resulting "dangling" vertices and edges (i.e., those without any adjacent faces), we are left with a quad mesh of black quads only. Note that we consider an edge is adjacent to a face if and only if that face has the edge as one of its sides. This *black-quad mesh* of the binary image may have zero or more connected components. Note that we consider two black quads sharing just a single vertex (i.e.,

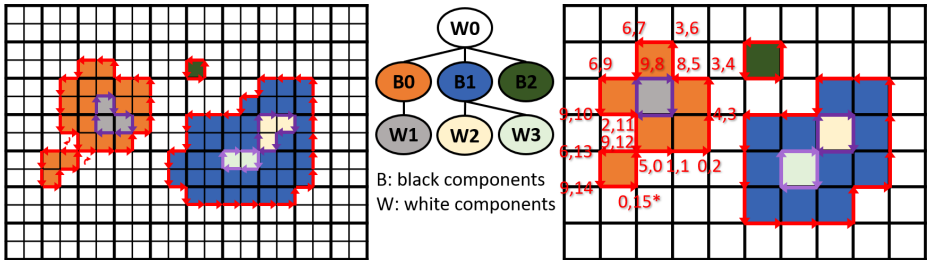

Fig. 3: Left: An input binary image of 3 black components (orange, blue, and green) and 4 white components (white, grey, goose-yellow, and light green). Middle: Its region adjacency graph (RAG). Right: a topologically correct downsampled solution with the same RAG. We draw the outer and inner boundaries of black components in red and purples, respectively. Observe that a black components' outer boundary can self-intersect. For inner boundaries, one cannot intersect itself but different ones can intersect with each other. We also show the type-index and distance of each corner on the first component's outer boundary. The corner with the "last" flag on is asterisked.

in a diagonal position) as neighbors. In each of the connected component, there may be zero or more holes. We denote the numbers of connected components and the total numbers of holes, namely the zeroth and first Betti numbers of the black-quad mesh, as the *topology* of the binary image.

We can instead remove all the black quads and the resulting dangling vertices and edges from the binary image, and have a *white-quad mesh* of the binary image. However, different to the rule for black quads, we consider two white quads to be neighbors if and only if they share an edge. In other words, *black quads use the 8-neighborhood rule but white quads use the 4-neighborhood rule.* With the black-quad mesh and white-quad mesh of a binary image both built, we can now partition the whole quad mesh into black-quad connected components ("black components" in short) and white components that together fully cover the whole quad mesh in a non-overlapping manner. We also build the *region adjacency graph (RAG)* of the binary image, which is a graph in which each node represents a component, and two nodes are connected by an edge if and only if the two components are adjacent ([19]). Note that every pair of adjacent components must be one black and one white. See Figure 3 for an example.

We now introduce the key concept behind our topology-preserving constraint formulation - the *boundaries* of connected components. We use notions of standard half-edge mesh data structures [1] .

Definition 1. *A boundary of a connected component's is a closed loop of consecutive half-edges such that every such half-edge's face belong to the component and its opposite half-edge's face does not not belong to the component.*

A boundary is *outer* if it is not of a hole. Otherwise, it is an *inner* boundary. Note that the consecutive half-edges of a outer and inner boundary go in the counter-clockwise and clockwise directions, respectively.

The case of the half-edges on the boundary of the whole quad mesh creates some confusion (because the face pointers of the opposites of these half-edges are null). To simplify discussions, we assume the faces on the boundary of the whole quad mesh are all white, and ignore the outer boundary of the white component that touches the quad mesh boundary (e.g., Figure 3). To support cases where there exist black faces touching the binary image boundary, we temporarily add a ring of white faces along the image boundary and later crop out the resulting white big-pixels. We now show an important lemma:

Lemma 1. *Faces of the opposite of all the half-edges in a boundary all belong to the same component.*

We show the proof in Supplementary Materials.

Lemma 1 implies that for every pair of adjacent black and white component as defined in the RAG of the original binary image, there exist exactly one closed loop of consecutive half-edges in counter-clockwise order (which is the outer boundary of the black component) that separates them.

3.1 Optimization Problem Formulation

We now formulate an integer programming (IP) optimization problem such that every solution constitutes a topologically correct downsampling result. We found that simply denoting the black-or-white decision of every big-pixels by a Boolean variable cannot cope with the complexity of the problem. Instead, we have to encode the *per-big pixel coverage by each components*. We first denote all components in the original binary image as C_i, $0 \leq i < N_c$, N_c is the number of components (both black and white). Now, for each component C_i, we collect all big-pixels that can potentially cover its pixels. The default way is that a pixel can be covered by a big-pixel if it is within the A-by-B block of pixels of the big-pixel. We extend this idea to allow a big-pixel to cover its *nearby* pixels outside its block by a distance threshold in the original resolution. Specially, we say a big-pixel at coordinate (X, Y) in the downsampled image, $0 \leq X < W/A$ and $0 \leq Y < H/B$, can cover a pixel at coordinate (x, y) in the original image if:

$$\begin{aligned} X * A - dx \leq x \leq ((X+1) * A - 1) + dx, \\ Y * B - dy \leq y \leq ((Y+1) * B - 1) + dy. \end{aligned} \tag{1}$$

dx and dy are distance thresholds in the two directions. We use $\lfloor A/4 \rfloor$ for dx and $\lfloor B/4 \rfloor$ for dy in our implementation.

Boolean Variables Declaration. We denote the collected set of big-pixels that can cover component C_i as:

$$P_j^i, \tag{2}$$

$0 \leq j < N_i$, N_i is the number of big-pixels that can cover C_i. For every P_j^i, we create a Boolean variable of the same name. We say P_j^i is the j-th covering big-pixel candidate for C_i. Note that a big-pixel can be a covering candidate for

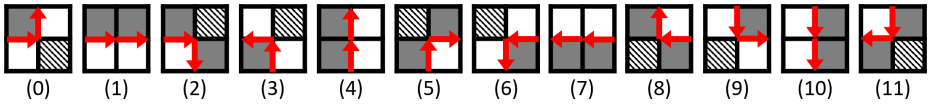

Fig. 4: The 12 possible corner configurations and their type-indices. Each has one incoming half-edge and one outgoing half-edge, one or two left hand-side (LHS) big-pixels (black), and one or two right hand-side (RHS) big-pixels (white) defined. Each configuration has a "pointing-to" position defined, e.g., for the 0-th it is on the top.

multiple components. We now formulate the four sets of constraints for the IP problem as follows.

1) Full and Non-Overlapping Coverage Constraint. Every big-pixel has to be covered by exactly one component. This means:

$$\forall_{(X,Y)} \sum_{i,j} P_j^i(X,Y) = 1. \tag{3}$$

$P_j^i(X,Y)$ denotes any P_j^i at coordinate (X,Y) in the downsampled binary image.

2) Component Non-Emptiness Constraint. Every component has to be covered by at least one big-pixel otherwise it would disappear. That is:

$$\forall_i \sum_{j}^{N_i} P_j^i \geq 1. \tag{4}$$

3) Local Neighborhood Correctness Constraint. Topological correctness in the neighborhood of each big-pixel is guaranteed by ensuring that two different black components can not be adjacent to each other by the 8-neighborhood rule and two different white components cannot be adjacent to each other by the 4-neighborhood rule. Accordingly, we collect all pairs of mutually incompatible big-pixel candidate Boolean variables (for example, two adjacent big-pixel candidates that belong to different black or white components) as (P_0^m, P_1^m), $0 \leq m < N_m$, N_m is the number of such pairs. We then have:

$$\forall_m \ (P_0^m + P_1^m) \leq 1. \tag{5}$$

Next, we introduce a constraint to ensure that, for every pair of adjacent black and white components as defined in the original RAG, there exists *exactly one closed loop of consecutive half-edges that separate the two components.*

4) Boundary-Preservation Constraint. Our main idea is to encode a solved boundary as one closed loop of consecutive *corners* in a downsampled binary image. A corner is two consecutive half-edges, $v0$-to-$v1$ and $v1$-to-$v2$, that intersect at $v1$. There exist exactly 12 possible *corner configurations*, as shown in Figure 4. Note that up to two corners can lie on the same vertex in a non-overlapping nor crossing-over manner (e.g., 5-th can collocate with 11-th, but not 4-th).

For every boundary between a black component Cb and a white component Cw, correctness in terms of local connectivity is ensured as follows. Observe that

each corner configuration has one or two adjacent "left hand-side (LHS)" big-pixels and one or two "right hand-side (RHS)" big-pixels defined (see Figure 4). We then require that a corner is part of a boundary *if and only if all its RHS big-pixels exist in Cb and all its LHS big-pixels exist in Cw*. This is realized by adding the following constraints for every possible corner candidate, momently denoted as a Boolean variable $Corner$:

$$0 \leq (\sum P_{Cb}^{RHS} + \sum P_{Cw}^{LHS}) - N * Corner \leq N - 1. \tag{6}$$

P_{Cb}^{RHS} are all possible big-pixel Boolean variables belong to black component Cb at the corner's RHS side, and P_{Cw}^{LHS} are all possible big-pixel Boolean variables belong to white component Cw at the corner's LHS side. N equals to sum of the number of all possible P_{Cb}^{RHS} and the number of all possible P_{Cw}^{LHS}.

In practice, for every boundary, we exhaustively search all 12 possible corner configurations on every possible inner vertices of the downsampled binary image, and enumerate all possible "corner candidates" as Boolean variables for the boundary as the ones that have both non-empty P_{Cb}^{RHS} and P_{Cw}^{LHS} sets. Equations 6 are then applied to every enumerated corner candidates of the boundary.

However, the above constraints still don't guarantee that there exists only *one* close loop of corners. Inspired by the "no-island" constraint proposed in [17], we further create a "distance" integer variable and a "last" Boolean variable to pair with every enumerated corner candidates of a boundary. We denote the former as $Dist_k$ and the latter as $Last_k$ for corner candidate variable $Corner_k$, $0 \leq k < N^{cc}$, N^{cc} is the number of corner candidates of the boundary. Here, our goal is to enforce that *distance values of consecutive corners along a boundary must be monotonically increasing*, with exactly one exception at the corner with the "last" flag on. This effectively ruled out any possibly of having multiple closed loops as each loop needs at least one "last" flag to be true. We now have:

$$\forall_k \ (Corner_k - ((\sum_l Dist_l) - Dist_k) - Last_k * \mathrm{BIG}) \leq 0, \tag{7}$$

where $Dist_l$ are distance value variables of compatible corners at $Corner_k$'s pointing-to position. BIG is a per-boundary integer constant that is guaranteed to be bigger than the largest possible length of the downsampled boundary (e.g., set to the length of the boundary in the original binary image). We also require that a corner's distance value must be zero if the corner itself is false:

$$\forall_k \ Dist_k \leq BIG * Corner_k. \tag{8}$$

Finally, we require that every boundary has exactly one "last" flag set to true:

$$\sum_{k \in \mathrm{Boundary}} Last_k = 1. \tag{9}$$

See Figure 3 right for an example of a solved boundary.

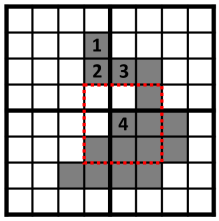

Fig. 5: We downsample a binary image of one black component and one white component by factor 4 in width and height. For the upper-left big-pixel of the black component, it's score value is 16, which is contributed by the four numbered black pixels with scores 6, 6, 3, and 1, respectively. To see this, the 4-th pixel's window is shown in red. It overlaps with the big-pixel at one pixel, therefore contributing 1 to the big-pixel's score.

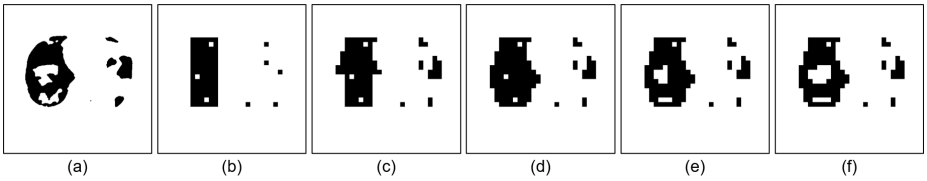

Fig. 6: Our baseline dilation-based method. (a) An input binary image. (b) The initial downsampled image. Each component is presented as a dot or a bounding box of inner holes. (c) After 10 iterations of black component dilation. (d) After black component dilation converged. (e) After 5 iterations of white component dilation. (f) Final result.

In summary, the above four sets of constraints have ensured that a solved binary image, which is the summation of all the active big-pixel variables, shall have the same RAG as the original. A proof is in the Supplementary Materials. Finally, we design our cost function as follows.

Objective Function. Our goal is to assign every big-pixel candidates of every component (i.e., P_j^i as defined in Equation 2) a "score" value, S_j^i, and formulate the objective function as to maximize the sum of scores of active big-pixels:

$$\max_{P_j^i} \sum_{i,j} (S_j^i * P_j^i), \tag{10}$$

where i denote the component index and j denotes the big-pixel candidate index in a component. Our general idea is to raise a score whenever an original pixel of a particular component is presented by an active big-pixel of the same component, and the raise is inversely proportional to the distance in between. We propose a way to achieve this as follows. For every pixel of component C_i at coordinate (x, y), we enumerate all its nearby pixel coordinates within a $(2 * dx + 1)$-by-$(2 * dy + 1)$ window (dx and dy are coverage distance thresholds used in Equation 1). Next, for every pixel in the window, we accumulate 1 to the score of the corresponding big-pixel candidate. See Figure 5 for an example.

We provide an analysis on the all possible outcomes of solving the IP problems in the Supplementary Materials.

4 A Baseline Dilation-Based Method

In short, we first enumerate all the black and white components in the input binary image. Next, in the downsampled image (initialized as all white), for

every component, we create a trivial and topologically correct representation of it. We do this by simply creating a dot at the center for every component. For components with holes, we create a bounding box of inner dots. Next, we iteratively *dilate* first all the black components then all the white components one pixel by one pixel. It becomes trivial to preserve topology for each dilation (i.e., skip ones that break or merge components). We stop until no more dilation operations can be found, or a time limit is reached. See Figure 6 for an example.

5 Results and Applications

We implemented our method in C++ and used Gurobi 11.0 [8] to solve the IP problems. We tested on a Windows PC with AMD 7 Ryzen 5800X CPU, 32GB RAMs, and NVIDIA RTX 3060 GPU. In Section 5.1, we tested our method versus other downsampling methods on the segmentation masks of a medical image dataset. We show results on more kinds of segmentation masks in the Supplementary Materials. In Section 5.2, we show that the speeds of two important binary image operations - persistent homology (PH) computation and shortest path calculation, can be significantly improved with small impact to the accuracy via straightforward applications of our method.

We re-implemented the ACN [6] method in C++ as the authors did not provide codes. Since ACN only does factor-2 downsampling, we run it multiple times to achieve downsampling at factors greater than 2. We use IoU and Dice scores of the foreground (black) components to measure pixel-wise similarity between two binary images. We use **Betti number error**, which measures sum of absolute errors of the zeroth and first Betti numbers, and **PH distance**, which measures the bottleneck distances of persistent diagrams [13] (computed by [25]), to measure topological similarities.

5.1 Medical Image Segmentation Masks Downsampling

We have found that visualizing the segmentation masks in medical datasets in reduced resolutions are helpful for users to identify intricate details such as tiny components or holes, and components that are very close-by but actually are separated. This is because small details are now represented by bigger pixels. Inspired by this, we tested several downsampling methods on the 542 segmentation masks (resolution 512x512) of lung coronavirus in the China National Center for Bioinformation (CNCB) dataset [7]. We show quantitative results in terms of topological accuracy, pixel-wise similarity, and speed in Table 1. We see that our method generated completely topologically correct results. Our results also have the lowest PH distances to the original among all methods. Remarkably, our results also achieved nearly the same levels of pixel-wise similarity as the non topology-preserving methods. The baseline dilation-based method also produced completely topologically correct results, but the pixel-wise accuracy metrics are much worse. Note that the computational costs of our methods are inversely proportional to the downsampling rates - this is because the higher

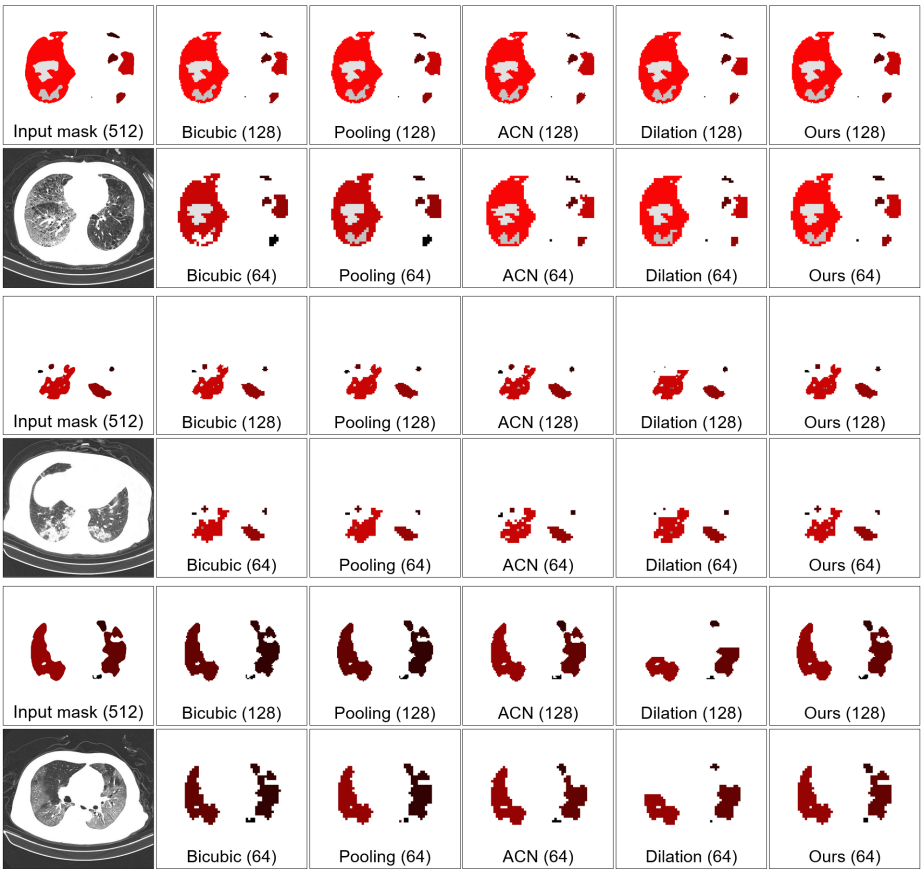

Fig. 7: Visualization of segmentation masks in downsampled resolutions. Details such as tiny components and holes are made easier to observe with bigger pixels. We deploy a coloring scheme to highlight the fact that non topology-preserving methods may produce results with erased small components/holes and merged/broken components.

the downsampling the fewer the big-pixel candidates. We consider our method's speed to be usable in interactive applications (less than 1 seconds for factor ≥ 4 cases). Our method cannot find feasible solutions for a few cases of the masks - at downsampling factor 2, 4, 8, and 16, there are 2, 1, 3, and 8 infeasible cases, respectively. Infeasible cases are generally detected instantly by our IP solver (times are included in our average time computations).

We show side-by-side qualitative comparisons in Figure 7. As a mean for visualization, we show different foreground and background components in different colors (levels of red and white/grey for foreground and background components, respectively). This coloring scheme helps to highlight an important fact that non topology-preserving downsampling methods may merge separate components (or holes) into one, or break one component or hole into multiple ones.

Table 1: Quantitative and speed comparisons of different downsampling methods on the 542 segmentation masks in the CNCB dataset [7] to different sizes. Best and second-best results are marked in red and blue, respectively.

	512x512 to 256x256 (factor 2)					512x512 to 128x128 (factor 4)				
Method	↑ IoU	↑ Dice	↓ Betti num. error	↓ PH distance	↓ Avg. time (s)	↑ IoU	↑ Dice	↓ Betti num. error	↓ PH distance	↓ Avg. time (s)
Bicubic	93.08%	96.34%	0.061	0.018	0.002	85.54%	91.78%	0.133	0.034	0.002
Pooling	93.41%	96.53%	0.046	0.015	0.615	85.99%	92.03%	0.144	0.046	0.160
ACN [6]	91.78%	95.61%	0.092	0.051	0.161	78.78%	87.52%	0.135	0.085	0.256
Dilation	59.71%	73.78%	0	0.026	2.348	61.69%	75.39%	0	0.028	0.544
Ours	93.10%	96.34%	0	0.005	1.976	85.12%	91.55%	0	0.012	0.819

	512x512 to 64x64 (factor 8)					512x512 to 32x32 (factor 16)				
Method	↑ IoU	↑ Dice	↓ Betti num. error	↓ PH distance	↓ Avg. time (s)	↑ IoU	↑ Dice	↓ Betti num. error	↓ PH distance	↓ Avg. time (s)
Bicubic	73.27%	83.46%	0.339	0.091	0.002	55.37%	68.28%	0.779	0.184	0.002
Pooling	73.33%	83.21%	0.404	0.115	0.046	54.35%	66.75%	0.969	0.214	0.017
ACN	60.81%	74.37%	0.356	0.150	0.335	41.00%	56.42%	0.600	0.181	0.414
Dilation	53.40%	68.07%	0	0.051	0.137	35.96%	50.49%	0	0.085	0.043
Ours	73.31%	83.55%	0	0.030	0.612	54.85%	68.92%	0	0.062	0.563

Table 2: Average times and PH distances of PH computations at different downsampled resolutions, tested on the CNCB dataset.

Image sizes:	512 (original)	256 (factor-2)	128 (factor-4)	64 (factor-8)	32 (factor-16)
Avg. time (s)	0.965	2.157	0.864	0.624	0.568
PH distance	-	0.005	0.012	0.030	0.062

5.2 Binary Image Operations Speed Improvement

Persistent Homology (PH) Computation Speed-up. As shown in Table 1, the quantitative differences of persistent diagrams (i.e., PH distances) between our downsampled results and the original are in general very small. A visual comparison of persistent diagrams, shown in the Supplementary Materials, also confirmed this. Therefore, one possible approach to speed up PH computation is by substituting the original binary images by smaller, downsampled versions computed by our method. For this approach to work, the time needed by our method to compute a downsampled image has to be shorter than the time saved by computing PH on a smaller (downsampled) image than the original. Our timing comparisons (shown in Table 2) using PH computation done by a commonly used tool [25] confirmed that this is the case when downsampling factor is ≥ 4.

Shortest Path Speed-up. Dijkstra shortest path computation in the black components is an operation that may take place if a binary image is taken as a geographical map (e.g., black components as land masses) in game design. We can speed up the computation by leveraging binary image downsampling methods. Our approach is as follows. Assume we have an original image, A, and an downsampled image, B. Our goal is to calculate a shortest path from pixel $p0$ to pixel $p1$ (both black pixels) in A. We now find the *corresponding* big-pixels of $p0$ and $p1$ in B, denoted as $P0$ and $P1$. This is done by: 1) identify the pixel p's component index. 2) Check if the big-pixel at p's corresponding coordinate

Table 3: In average, conducting 200 Dijkstra shortest path computations with random start and end pixels needs about 37.67 seconds in a 512x512 binary image. We demonstrate that doing so in downsampled images can have significant time saving with small impact to the accuracy. We show the average shortest distance errors, numbers of false positives (FP) and false negatives (FN), and total time (200 computations) using different downsampling methods at different sizes. Only using our method and the dilation method is free of FP and FN cases, while ours has smaller distance errors.

Avg. dist. error, FP, FN| time(s)	128 (factor-4)		64 (factor-8)		32 (factor-16)	
Bicubic	1.65, 0.00, 0.78|	2.49	5.05, 0.00, 2.12|	0.95	11.57, 0.00, 1.85|	0.96
Pooling	2.63, 0.00, 0.00|	2.96	6.60, 0.01, 0.26|	1.21	14.64, 0.01, 0.40|	1.17
ACN [6]	1.80, 0.01, 0.10|	3.22	5.82, 0.00, 0.29|	1.92	13.44, 0.05, 0.47|	2.00
Dilation	0.38, 0.00, 0.00|	3.66	1.00, 0.00, 0.00|	1.48	2.52, 0.00, 0.00|	1.27
Ours	0.17, 0.00, 0.00|	3.63	0.19, 0.00, 0.00|	1.34	0.47, 0.00, 0.00|	0.99

in B is belong to the same component. 3) If yes, use the big-pixel. 4) If not, find the closest big-pixel (to big-pixel boundaries in the original resolution) with the same component index. We then take the computed shortest path from $P0$ to $P1$ in B, scaled back to the original resolution, as the approximated shortest path for $p0$ to $p1$. Note that in this application, using non topologically-correct downsampled images is unsuitable: it is because when accurate component indices of big-pixels are unavailable, we can only find each pixel's corresponding big-pixels approximately by distance. This may cause originally unreachable pairs of points (i.e., in different components) to become reachable in the downsampled image ("false positive" cases) and originally reachable pairs of points to become unreachable ("false negative" cases).

We conduct experiments by randomly choosing 200 pairs of starting and ending pixels in each of the 542 binary images in the CNCB dataset. We use Boost Graph Library [21]'s Dijkstra implementation. Note that we allow choosing two pixels in different black components (no paths in between). We show testing results in Table 3. Note that time saving happens because typically shortest path computation are done many times in a binary image, and total time saving will be greater than the time spent on doing downsampling once per image.

6 Conclusion and Future Work

In summary, our method generates downsampled binary images that are guaranteed to have the same RAG as the original. This is in fact a stronger guarantee then just having the same Betti numbers - as two 2D graphs with the same RAG must have the same Betti numbers, but the inverse is not necessarily true (e.g., same number of holes but appear in different connected components).

A main limitation is to handle binary images with very thin structures such as road networks and blood vessels. Currently, such structures may not be satisfactorily preserved because our objective function design preserves black and white components with the same preference. Therefore, an interesting research direction may be to design alternative cost functions to handle such binary images. Codes are available at: github.com/pengchihan/BinaryImageDownsampling.

Acknowledgements. This work is funded by the National Science and Technology Council of Taiwan (project number 111R10286C).

References

1. Cgal 5.6 - halfedge data structures, https://doc.cgal.org/latest/HalfedgeDS/index.html
2. Abu-Ain, W., Abdullah, S.N.H.S., Bataineh, B., Abu-Ain, T., Omar, K.: Skeletonization algorithm for binary images. Procedia Technology **11**, 704–709 (2013)
3. Boudaoud, L.B., Sider, A., Tari, A.: A new thinning algorithm for binary images. In: 2015 3rd international conference on control, engineering & information technology (CEIT). pp. 1–6. IEEE (2015)
4. Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)
5. Clough, J.R., Byrne, N., Oksuz, I., Zimmer, V.A., Schnabel, J.A., King, A.P.: A topological loss function for deep-learning based image segmentation using persistent homology. IEEE transactions on pattern analysis and machine intelligence **44**(12), 8766–8778 (2020)
6. Decencière, E., Bilodeau, M.: Adaptive crossing numbers and their application to binary downsampling. Image Analysis & Stereology **26**(2), 73–81 (2007)
7. Gong, Z., Zhu, J., Li, C., Jiang, S., Ma, L., Tang, B., Zou, D., Chen, M., Sun, Y., Song, S., Zhang, Z., Xiao, J., Xue, Y., Bao, Y., Du, Z., Zhao, W.: An online coronavirus analysis platform from the national genomics data center. Zoological Research **41**(6), 705–708 (2020)
8. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023), https://www.gurobi.com
9. Hu, X.: Structure-aware image segmentation with homotopy warping. Advances in Neural Information Processing Systems **35**, 24046–24059 (2022)
10. Hu, X., Li, F., Samaras, D., Chen, C.: Topology-preserving deep image segmentation. Advances in neural information processing systems **32** (2019)
11. Jain, R., Kasturi, R., Schunck, B.: Machine Vision. Computer science series, McGraw-Hill (1995)
12. Li, L., Ma, Q., Ouyang, C., Li, Z., Meng, Q., Zhang, W., Qiao, M., Kyriakopoulou, V., Hajnal, J.V., Rueckert, D., et al.: Robust segmentation via topology violation detection and feature synthesis. In: International Conference on Medical Image Computing and Computer-Assisted Intervention. pp. 67–77. Springer (2023)
13. Maria, C., Boissonnat, J.D., Glisse, M., Yvinec, M.: The gudhi library: Simplicial complexes and persistent homology. In: Mathematical Software–ICMS 2014: 4th International Congress, August 2014. Proceedings 4. pp. 167–174. Springer (2014), https://gudhi.inria.fr/doc/latest/group__bottleneck__distance.html
14. Mosinska, A., Marquez-Neila, P., Koziński, M., Fua, P.: Beyond the pixel-wise loss for topology-aware delineation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3136–3145 (2018)
15. Ngo, P., Passat, N., Kenmochi, Y., Talbot, H.: Topology-preserving rigid transformation of 2d digital images. IEEE Transactions on Image Processing **23**(2), 885–897 (2014). https://doi.org/10.1109/TIP.2013.2295751
16. Passat, N., Ngo, P., Kenmochi, Y., Talbot, H.: Homotopic affine transformations in the 2D Cartesian grid. Journal of Mathematical Imaging and Vision **64**(7), 786–806 (2022). https://doi.org/10.1007/s10851-022-01094-y

17. Peng, C.H., Yang, Y.L., Bao, F., Fink, D., Yan, D.M., Wonka, P., Mitra, N.J.: Computational network design from functional specifications. ACM Trans. Graph. **35**(4) (jul 2016)
18. Phuc Ngo, Yukiko Kenmochi, N.P., Talbot, H.: Topology-preserving conditions for 2d digital images under rigid transformations. Journal of Mathematical Imaging and Vision **49**, 418–433 (2014). https://doi.org/10.1007/s10851-013-0474-z
19. Shapiro, L.G., Stockman, G.C.: Computer Vision. Pearson (2001)
20. Shit, S., Paetzold, J.C., Sekuboyina, A., Ezhov, I., Unger, A., Zhylka, A., Pluim, J.P., Bauer, U., Menze, B.H.: cldice-a novel topology-preserving loss function for tubular structure segmentation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16560–16569 (2021)
21. Siek, J., Lee, L.Q., Lumsdaine, A.: Boost graph library. http://www.boost.org/libs/graph/ (June 2000)
22. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015)
23. Strutz, T.: The distance transform and its computation. arXiv preprint arXiv:2106.03503 (2021)
24. Suzuki, S., et al.: Topological structural analysis of digitized binary images by border following. Computer vision, graphics, and image processing **30**(1), 32–46 (1985)
25. Tauzin, G., Lupo, U., Tunstall, L., Pérez, J.B., Caorsi, M., Medina-Mardones, A.M., Dassatti, A., Hess, K.: giotto-tda: A topological data analysis toolkit for machine learning and data exploration. Journal of Machine Learning Research **22**(39), 1–6 (2021), http://jmlr.org/papers/v22/20-325.html
26. Wang, J., Kosinka, J., Telea, A.: Spline-based medial axis transform representation of binary images. Computers & Graphics **98**, 165–176 (2021)
27. Yokoi, S., Toriwaki, J.I., Fukumura, T.: An analysis of topological properties of digitized binary pictures using local features. Computer Graphics and Image Processing **4**(1), 63–73 (1975)