EUROGRAPHICS Workshop on ... '0x
N.N. and N.N.
(Editors)

Volume 0 (1981), Number 0

Topological Simplification of Nested Shapes

D. Zeng[1] and E. Chambers[2] and D. Letscher[2] and T. Ju[1]

[1]Washington University in St. Louis, USA
[2]Saint Louis University, USA

Abstract

We present a method for removing unwanted topological features (e.g., islands, handles, cavities) from a sequence of shapes where each shape is nested in the next. Such sequences can be found in nature, such as a multi-layered material or a growing plant root. Existing topology simplification methods are designed for single shapes, and applying them independently to shapes in a sequence may lose the nesting property. We formulate the nesting-constrained simplification task as an optimal labelling problem on a set of candidate shape deletions ("cuts") and additions ("fills"). We explored several optimization strategies, including a greedy heuristic that sequentially propagates labels, a state-space search algorithm that is provably optimal, and a beam-search variant with controllable complexity. Evaluation on synthetic and real-world data shows that our method is as effective as single-shape simplification methods in reducing topological complexity and minimizing geometric changes, and it additionally ensures nesting. Also, the beam-search strategy is found to strike the best balance between optimality and efficiency.

CCS Concepts

• ***Computing methodologies*** → ***Shape analysis; Volumetric models;***

1. Introduction

Topological simplification is the problem of reducing unwanted topological features from 3D shapes. These features include islands (i.e., connected components), cavities (i.e., voids inside the shape), and handles (i.e., "loops"). These artifacts are typically resulted from reconstructing the shape from noisy or incomplete raw inputs, such as images or point clouds. Without simplification, spurious topological features may present significant challenges for many geometry processing tasks such as mesh simplification, surface parameterization, shape matching, and physical simulations.

Many methods have been proposed to simplify the topology of a single 3D solid shape (see review in Section 2.1). However, for shapes that exist in a collection, the processing of individual shapes may be subject to additional constraints among the shapes. In this work, we consider a sequence of *nested* solid shapes, such that each shape is a subset of the next shape in the sequence. One example of nested shapes is the time-series of an expanding structure, such as the growing roots of a plant, where the shape at each time-point is strictly a subset of the shape at the next time-point. Another example is a multi-layered material, such as a geological model consisting of multiple strata, a mechanical part with several coatings, or a biological structure with several tissues. Given a material composed of n layers, taking the union of the k inner-most layers, as k increases from 1 to n, creates a sequence of nested solid shapes. For such shape sequences, maintaining the nesting relations between successive shapes is important for downstream analysis, whether it is physical simulation on a layered material or computing the time function of an expanding structure.

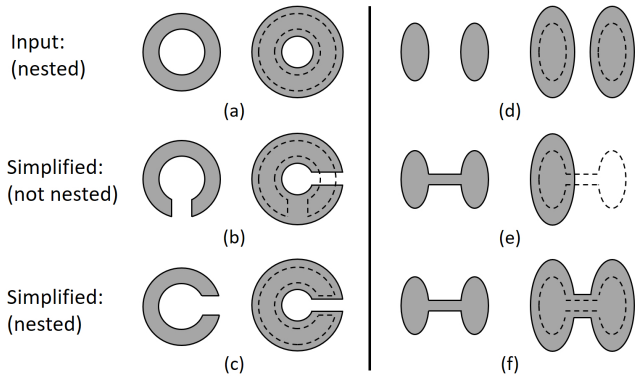

Figure 1: *Two examples (left and right) of a pair of nested shapes containing topological features (top), and after topological simplification that violates (middle) or preserves (bottom) nesting. The outline of the first shape in each pair is overlayed on the second.*

Performing topological simplification independently on each shape in a nesting sequence could result in shapes that are no longer nested. For example, a handle may be cut in different locations in two consecutive shapes (Figure 1 (b)), whereas two islands may be bridged in one shape but one of them is removed in the next shape (Figure 1 (e)). Such violations of nesting are often found on

real-world data, as shown in Figure 2 (middle row) which applies a recent single-shape topology simplification method [ZCLJ20] to a time-series of growing pennycress roots.

An alternative approach that guarantees nesting is to employ methods that analyze and simplify the topological features of scalar fields (see review in Section 2.2). Specifically, one may create a scalar field whose level sets are the input shape sequence, simplify the topology of the scalar field, and extract the level sets afterwards. However, existing field-based simplification methods are limited to removing extrema (maxima and minima), which correspond to islands and cavities in the level sets, and thus are not able to effectively remove topological handles in 3D (see Figure 13).

In this paper, we propose a novel topological simplification method designed for nested shapes. Our method simplifies all three types of topological features (islands, handles, cavities) on each shape while maintaining their nesting relation. Our method builds upon an existing single-shape simplification method [ZCLJ20], which adopts a global optimization approach to maximally simplify topology while minimizing geometric changes. We make two main technical contributions:

- We formulate nested simplification as a graph labelling problem (Section 4). Similar to [ZCLJ20], our formulation makes use of a set of pre-computed, candidate modifications to each shape, known as *cuts* (deletions from the shape) and *fills* (additions to the shape). Our formulation extends the one in [ZCLJ20], which concerns a single shape, by considering multiple shapes while imposing nesting constraints *between* shapes. Additionally, we propose to compute *nesting-aware* cuts and fills that locally respect the nesting constraints.
- We propose several strategies for solving the graph labelling problem for a given set of nesting-aware cuts and fills (Section 5). These include a greedy heuristic that propagates solutions from one shape to the next, a state-space search algorithm that is provably optimal, and a beam-search variant that trades off optimality for efficiency.

We tested our method on both synthetic and real-world shape sequences (Figure 2 and more in Section 6). We found that all optimization strategies are effective in reducing topological complexity of individual shapes while maintaining nesting. On complex data, we found that, among the three optimization strategies, the beam-search algorithm strikes the best balance between solution quality and performance.

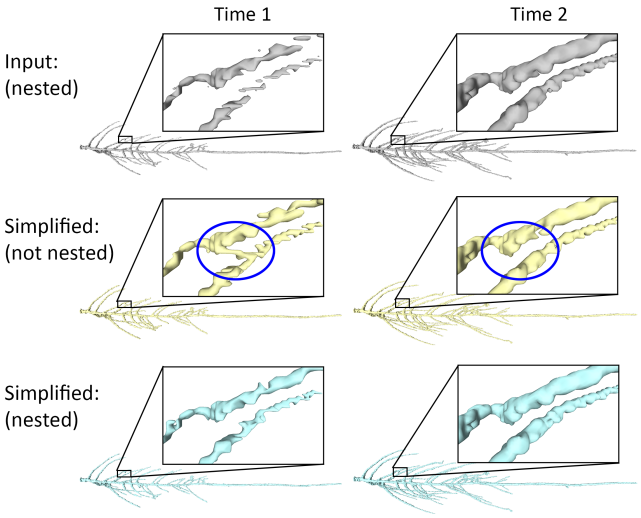

Figure 2: *Simplifying the topology of a growing sequence of pennycress root (top row, showing two time points) by applying the simplification method [ZCLJ20] to each shape independently (middle row) and by our proposed method (bottom row). While both methods fully simplify the topology of each shape, the results of [ZCLJ20] are no longer nested (e.g., regions highlighted in blue circles) but ours are.*

2. Related Work

2.1. Topological simplification of shapes

Topological simplification of 3D shapes has been extensively studied in the past. While the general problem of maximal simplification within an error bound is known to be NP-hard [ABD*15], many practical heuristics have been proposed. Some methods only remove topological handles, for which a common approach is computing the Minimum Spanning Tree on a graph [SL01, CW06, HXBNP02, ZJH07]. Methods to remove all three types of topological features (islands, handles, and cavities) include morphological opening and closing [NT03], inflation or deflation from a topologically simple *seed* [KG01, BK02, SV03], or tools from persistent homology [CJL*18]. However, these above methods either only add to, or only delete from, the shape to remove its topological features. This may lead to excessive geometric changes, since some topological features are either to be cut (e.g., a thin loop) whereas others are easier to be filled (e.g., a narrow tunnel).

Methods that allow both addition and deletion often apply some local heuristics to decide where to add or delete [KG01, WHDS04, SPF07, JZH07]. However, these heuristics do not consider the global optimality of the changes. The recent work of Zeng et al. [ZCLJ20] tackles topological simplification as a global optimization problem. After computing a set of candidate cuts and fills, this method formulates a graph labelling problem on these candidates that aims at maximally simplifying the topology while minimizing the amount of geometric changes. The problem is then solved by a transformation to the Node-Weighted Steiner Tree (NWST) problem.

While these methods can simplify individual shapes in a collection, they do not respect any notion of consistency (e.g., nesting) among the shapes. In this regard, our method makes a first step towards consistency-constrained topological simplification of a shape collection. Our method is an extension of [ZCLJ20] in two ways: (1) we extend its candidate-based formulation to multiple shapes under the nesting constraint, and (2) we use its label optimization algorithm as a building-block in our own optimization.

2.2. Topological simplification of scalar fields

Topological abstractions, such as the Morse-Smale complex and persistence diagram, are useful for analyzing scalar fields in Topological Data Analysis (TDA) [HLH*16]. Since practical data is often noise-ridden, such noise needs to be removed from a scalar field prior to analysis. Numerical methods [BHEP04, WGS10, PF09, GJR*14] simplify a scalar field by maximally removing its local extrema (minima and maxima) except for those in a given constraint set. Combinatorial methods [EMP06, BLW12, TP12, LGMT20] remove those extrema whose importance, given by some measure such as persistence [ELZ02], is below a threshold while maintaining an error bound.

A major limitation of scalar field simplification methods is that they are generally not effective in removing saddle points in a 3D scalar field. Since saddles are responsible for topological handles on the level sets, these methods have limited utility in solving our nesting-constrained simplification problem by representing the shape sequence as level sets (see Figure 13).

3. Background

Our algorithm represents and analyzes shapes represented by cells within a cell complex. We briefly review key concepts in cell complex topology relevant to the discussion of our method. Readers are referred to standard literature such as [Hat02] for in-depth discussions on the subject.

A k-dimensional cell, or k-cell, is an open set homeomorphic to an open k-dimensional ball. A set of disjoint cells is called a *cell complex* if, for each cell in the complex, its boundary is completely covered by other lower-dimensional cells in the complex. For example, a cell complex may consist of a cube (a 3-cell), its six bounding squares (2-cells), twelve edges (1-cells), and eight vertices (0-cells). A cell x is said to be a *face* of another cell y if x lies in the boundary of y.

Given a cell complex $\mathcal{C}$ covering a d-dimensional space, we denote the set of all d-cells of $\mathcal{C}$ as $\mathcal{C}_d$ and call them the *top-dimensional cells*. For example, if $\mathcal{C}$ is a cubical or tetrahedral decomposition of space, $\mathcal{C}_d$ are the cubes or tetrahedra (excluding their low-dimensional faces). Our algorithm represents a shape as a subset of $\mathcal{C}_d$. Strictly speaking, the top-dimensional cells are open sets and disjoint from each other. For the purpose of analyzing topology, we need to define a connected shape from them. Given a set of top-dimensional cells $O \subseteq \mathcal{C}_d$, we can define the shape as either their closure $\Omega(O)$ or as the complement of the closure of the remaining top-dimensional cells $\mathcal{C} \setminus \Omega(\mathcal{C}_d \setminus O)$. See a 2D illustration in Figure 3. In the case that $\mathcal{C}$ is a quadrilateral decomposition in 2D, these two ways of shape definition are equivalent respectively to the 8- and 4-connectivity in digital topology [KR04].

The topology of a shape can be characterized by its Betti numbers. Given a d-dimensional shape S, the k-th *Betti number* $\beta_k(S)$, for $k \geq 0$, is the rank of the k-th homology group on S. In 2-dimensions, $\beta_0(S)$ and $\beta_1(S)$ respectively measure the number of connected components and cavities of S. In 3-dimensions, $\beta_0(S), \beta_1(S), \beta_2(S)$ measure the number of connected components, handles, and cavities of S, respectively.

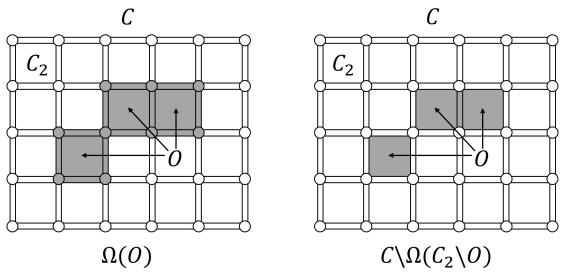

Figure 3: *Two ways of defining a shape (shaded cells) from three 2-cells O in a 2D cell complex $\mathcal{C}$. $\mathcal{C}_2$ is the set of all 2-cells in $\mathcal{C}$.*

A topological feature (e.g., connected component, handle or cavity) can be created or removed using one of the two types of topological surgeries, namely *cutting* (i.e., removing contents from the shape) and *filling* (i.e., adding contents to the shape). For example, a component can be removed by either deleting the entire component (i.e., cutting) or connecting it with another component via a bridge (i.e., filling). A handle can be removed by either breaking the handle ring (i.e. cutting) or filling the handle hole. A cavity can be removed by either connecting it with the exterior by a tunnel (i.e. cutting) or filling the entire cavity. These surgeries are illustrated in 3D in Figure 4.

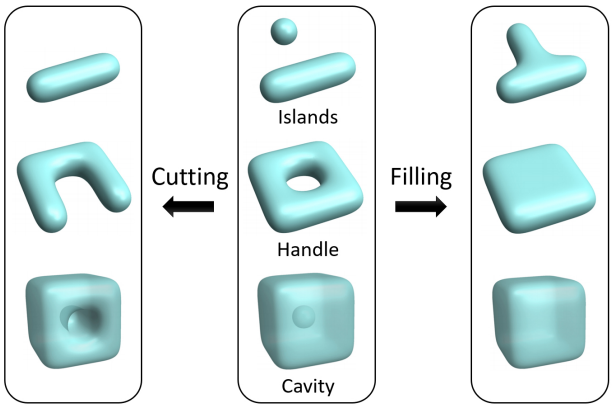

Figure 4: *Removing islands (top), handles (middle), and cavities (bottom) by cutting or filling. Adapted from Figure 2 of [ZCLJ20] and courtesy of the authors.*

4. Problem formulation

Given a nesting sequence of 2D or 3D shapes, our goal is to modify each shape in the sequence such that (1) the modified shape's topology is as simple as possible, (2) the modified shape differs from the original shape as little as possible, and (3) the modified shape sequence remains nesting.

We first present a general formulation of the problem, including a formal specification of the input and output (Section 4.1). To make the optimization task more trackable, we introduce a simplified formulation that considers only a pre-computed set of candidate

cuts and fills (Section 4.2). Next, we characterize candidates that are compatible with the nesting constraint (which we call *nesting-aware* candidates) and develop an algorithm for finding such candidates (Section 4.3). Finally, we tailor the problem formulation to nesting-aware candidates (Section 4.4).

4.1. General formulation

The main input to our method is a shape sequence $\{T_1,\dots,T_n\}$ defined on a common cell complex $\mathcal{C}$ in d-dimensional space for $d=2,3$. Each T_i is a subset of the top-dimensional cells (d-cells) of $\mathcal{C}$, that is, $T_i \subseteq \mathcal{C}_d$. The sequence is nesting in the sense that $T_i \subseteq T_{i+1}$ for all $i=1,\dots,n-1$. For simplicity, henceforward we shall use T_i to denote both the composing d-cells and the connected shape represented by them (using either definition in Figure 3).

To suit different application scenarios, our method can take in two additional and optional inputs from the user. First, the user may define a per-cell *geometric cost* $g_i(c)$ for each d-cell $c \in \mathcal{C}_d$ and each $i=1,\dots,n$. This cost measures the geometric change when c is added to or deleted from T_i. By default, $g_i(c)$ is set to be the area (in 2D) or volume (in 3D) of c, but it can be customized. For example, it can be weighted by an additional scalar field (e.g., confidence map or SDF) that may come with each shape. For notational simplicity, we write the sum of costs over a set of d-cells C as $g_i(C)$.

Secondly, the user may provide two additional sets of d-cells, a *kernel* K and a *neighborhood* N, such that $K \subseteq T_1$ and $T_n \subseteq N$. These two sets define an "envelope" within which the modified shape sequence will be constrained. Note that, if both K,N have non-trivial topology, the modified shapes will *preserve* those topological features that persist from K to N. If not provided by the user, K is set to be a single d-cell inside T_1 and $N=\mathcal{C}_d$. Since K has a simple topology (a single connected component without handles or cavities), our method will attempt to remove all topological features in the shape sequence.

Given the shape sequence $\{T_1,\dots,T_n\}$, the geometric cost functions g_i, the kernel K and neighborhood N, we seek a modified sequence of shapes $\{T_1',\dots,T_n'\}$ that minimizes, lexicographically, the following vector energy that measures (firstly) the total number of topological features and (secondly) the total geometric change,

$$E(\{T_1',\dots,T_n'\}) = \{\sum_{i=1}^{n}\sum_{k=0}^{d}\beta_k(T_i'),\ \sum_{i=1}^{n} g_i(T_i' \ominus T_i)\}, \tag{1}$$

where β_k is the k-th Betti number and $\ominus$ is the symmetric difference operator, subject to the constraint that the modified sequence is nesting and sandwiched within K,N:

$$K \subseteq T_i' \subseteq \dots \subseteq T_n' \subseteq N \tag{2}$$

4.2. Candidate-based formulation

The problem formulated above is NP-hard even when $n=1$ and without considering the geometric cost [ABD*15]. In the case of $n=1$, an alternative and more computationally friendly formulation was proposed in [ZCLJ20], which restricts the shape modifications to a pre-computed set of *candidate cuts and fills*. A candidate cut (resp. fill) is a group of d-cells in the original shape (resp. its complement) whose simultaneous deletion (resp. addition) removes one or more topological features of the shape. For example, a cut could be all cells that make up a connected component of the shape, whereas a fill could be a group of cells that form a bridge connecting two components. The energy E (Equation 1) can then be expressed as a function of a binary labelling on these candidates, where a label of 0 (resp. 1) means a candidate is included in (resp. excluded from) the shape.

We extend the candidate-based formulation of [ZCLJ20] to $n>1$ with the nesting constraint. We assume that there exists a set of candidate cuts and fills for each shape T_i, denoted as X_i (their computation will be discussed shortly). Given a 0/1 labelling L of the candidates, the modified shape, denoted by T_i^L, can be written as:

$$T_i^L = (T_i \setminus \mathbf{X}_i^{L,0}) \cup \mathbf{X}_i^{L,1} \tag{3}$$

where $\mathbf{X}_i^{L,0}$ and $\mathbf{X}_i^{L,1}$ are the unions of candidates of X_i that are labelled by L respectively as 0 and 1. If we replace the modified shapes T_i' in the general formulation (Equations 1,2) by T_i^L, the problem becomes seeking a labelling L that minimizes the energy

$$E(L) = E(\{T_1^L,\dots,T_n^L\}), \tag{4}$$

where the righthand side is defined in Equation 1, while satisfying the constraint that

$$K \subseteq T_i^L \subseteq \dots \subseteq T_n^L \subseteq N. \tag{5}$$

4.3. Nesting-aware candidates

The formulation above simplifies the problem by limiting the shape modifications to the use of the pre-computed candidates. The choice of the candidate cuts and fills therefore plays an important role in how well the solution of the simplified problem minimizes the original energy (Equation 1).

Naturally, we would like to have candidates that can maximally simplify each shape while having low geometric costs. Such candidates can be obtained, for each shape in the sequence, using the inflation/deflation approach in [ZCLJ20]. To compute the candidate cuts on a shape T (restricted to the envelop within a kernel K and a neighborhood N), their method expands K towards T by iteratively adding cells of T while preserving the topology of K. When no more cells can be added without incurring a topological change, each connected component of the difference between T and the expanded kernel becomes a cut[†]. The candidate fills are constructed in a symmetric fashion by maximally shrinking the neighborhood N towards T while preserving the topology of N and taking the connected components of the difference between the shrunken neighborhood and T. By construction, such cuts and fills are sufficient to fully simplify the topology of T (except for those topological features shared by both K and N). Furthermore, the order of cells being added or deleted is chosen so that the cuts and fills tend to consist of cells with low per-cell geometric costs.

However, candidates constructed independently for each shape

[†] The connected components are based on the so-called *R-connectivity* defined in [ZCLJ20] to ensure consistency with their graph formulation.

may not be useful towards satisfying the nesting constraint (Equation 5). Let $\mathbf{X}_i$ be the union of all candidates in X_i, and define the *per-shape kernel* as $K_i = T_i \setminus \mathbf{X}_i$ and the *per-shape neighborhood* as $N_i = T_i \cup \mathbf{X}_i$. Observe that, regardless of the labelling L, the modified shape T_i^L is sandwiched between K_i and N_i. Now, consider a candidate cut in the next shape T_{i+1} which shares some common cells with K_i (e.g., the cut c_3 in Figure 5 (a)). Deleting that cut from T_{i+1} will result in a shape that does not completely contain K_i and hence fails to contain T_i^L for any L. Symmetrically, if a candidate fill in the previous shape T_{i-1} contains some cells that are not in N_i (e.g., the fill f_1 in Figure 5 (b)), then adding the fill to T_{i-1} will result in a shape that is not completely contained within N_i and in turn fails to nest inside T_i^L for any L. In both cases, the cut or fill is "useless" in the sense that it cannot be applied without violating the nesting constraint.

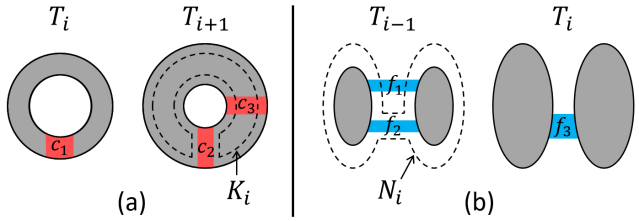

Figure 5: *Two examples (a,b) of pairs of consecutive shapes with candidate cuts (red) and fills (blue). In (a), the cut c_2 is nesting-aware because it is disjoint from the previous per-shape kernel K_i (shown in outline), but c_3 is not. In (b), the fill f_2 is nesting-aware because it is contained within the next per-shape neighborhood N_i (shown in outline), but f_1 is not.*

The discussion above shows that a candidate $x \in X_i$ is useful towards satisfying nesting only if it is disjoint from the previous per-shape kernel (i.e., $x \cap K_{i-1} = \emptyset$) and nested within the next per-shape neighborhood (i.e., $x \subseteq N_{i+1}$)[‡]. We call such candidates *nesting-aware*. For example, in Figure 5, the cut c_2 and fill f_2 are both nesting-aware, whereas the cut c_3 and fill f_1 are not.

To compute nesting-aware candidates, we first given an alternative characterization using per-shape kernels K_i and neighborhoods N_i:

Proposition 4.1 All candidates in $\{X_1, \ldots, X_n\}$ are nesting-aware if and only if

$$K \subseteq K_1 \subseteq \ldots \subseteq K_n \tag{6}$$

and

$$N_1 \subseteq \ldots \subseteq N_n \subseteq N \tag{7}$$

where $K_i = T_i \setminus \mathbf{X}_i$ and $N_i = T_i \cup \mathbf{X}_i$.

Proof We first show sufficiency. Let $K_0 = K$ and $N_{n+1} = N$. Since $K_{i-1} \subseteq K_i$ and $\mathbf{X}_i \cap K_i = \emptyset$ for $i = 1, \ldots, n$, $x \cap K_{i-1} = \emptyset$ for any $x \in X_i$. Similarly, Since $N_i \subseteq N_{i+1}$ and $\mathbf{X}_i \subseteq N_i$ for $i = 1, \ldots, n$, $x \subseteq N_{i+1}$ for any $x \in X_i$. To show necessity, suppose on the contrary that all candidates are nesting-aware but $K_{i-1} \nsubseteq K_i$ for some $i \in [1,n]$. Hence there exists some d-cell $c \in K_{i-1}$ but $c \notin K_i$. Since $K_{i-1} \subseteq T_{i-1} \subseteq T_i$, $c \in T_i$, and hence c must belong to some cut $x \in X_i$. However, x overlaps with K_{i-1} at c, which contradicts that x is nesting-aware. A similar contradiction can be reached if all candidates are nesting-aware but $N_i \nsubseteq N_{i+1}$ for some $i \in [1,n]$. □

The characterization leads to a variation of the inflation/deflation approach of [ZCLJ20] to compute nesting-aware candidates that are also sufficient for simplifying topology and low in geometric costs. To compute the candidate cuts, instead of expanding the same kernel K to every shape T_i, we expand the previous per-shape kernel K_{i-1} towards T_i while preserving the topology of K_{i-1} and guided by the geometric cost g_i. The result of maximal expansion becomes the current per-shape kernel K_i, and the connected components of $T_i \setminus K_i$ become the candidate cuts in X_i. Our algorithm starts from T_1 and iteratively creates a expanding sequence of per-shape kernels $\{K_1, \ldots, K_n\}$ satisfying Equation 6. The candidate fills are created in a symmetric fashion, this time working backwards from T_n. For each shape T_i, we shrink the next per-shape neighborhood N_{i+1} towards T_i to produce the current per-shape neighborhood N_i, and the connected components of $N_i \setminus T_i$ become the candidate fills in X_i. The process results in a sequence of shrinking per-shape neighborhoods $\{N_n, \ldots, N_1\}$ satisfying Equation 7. The inflation and deflation processes are illustrated in 2D in Figure 6.

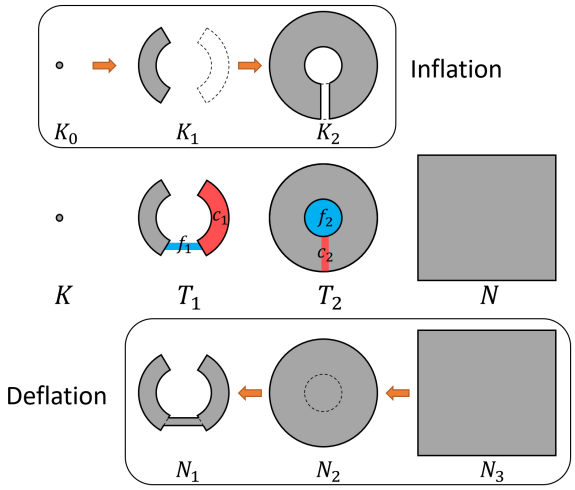

Figure 6: *Computing nesting-aware candidates. Middle row: An input sequence consisting of two shapes T_1, T_2, the kernel K, and neighborhood N. Top: Inflating $K_0 = K$ towards successive shapes in the sequence, while preserving topology, results in per-shape kernels K_1, K_2. Bottom row: Deflating $N_3 = N$ towards successive shapes in the reverse order, while preserving topology, results in per-shape neighborhoods N_2, N_1. Components of $T_i \setminus K_i$ and $N_i \setminus T_i$ become the candidate cuts (red) and fills (blue) in the middle row.*

4.4. Formulation based on nesting-aware candidates

The use of nesting-aware candidates further simplifies the problem formulation. The key observation, formally stated below, is that the inclusion constraint of Equation 5 can be replaced by labelling constraints on pairs of *overlapping* nesting-aware candidates:

‡ We assume $K_0 = K$ and $N_{n+1} = N$

Proposition 4.2 If all candidates $\{X_1,\dots,X_n\}$ are nesting-aware, then Equation 5 holds if and only if the following holds: for any $i=1,\dots,n-1$ and any two candidates $x\in X_i$ and $y\in X_{i+1}$ such that $x\cap y\neq\emptyset$, either $L(x)=0$ or $L(y)=1$.

Proof We first note that

$$T_i^L = K_i\cup\mathbf{X}_i^{L,1} = N_i\setminus\mathbf{X}_i^{L,0}.$$

Hence $T_i^L\subseteq T_{i+1}^L$ is equivalent to

$$K_i\cup\mathbf{X}_i^{L,1}\subseteq N_{i+1}\setminus\mathbf{X}_{i+1}^{L,0}.$$

Since all candidates are nesting-aware, and by Proposition 4.1, $K_i\subseteq N_i\subseteq N_{i+1}$. Hence the inequality above holds if and only if $\mathbf{X}_i^{L,1}\cap\mathbf{X}_{i+1}^{L,0}=\emptyset$. The latter, in turn, is equivalent to asking that, for any two candidates $x\in X_i$ and $y\in X_{i+1}$ such that $x\cap y\neq\emptyset$, $\{L(x),L(y)\}$ cannot be $\{1,0\}$. □

We call the simultaneous assignment of label 1 to x and 0 to y for a pair of overlapping candidates $x\in X_i$ and $y\in X_{i+1}$ a *conflict*. Note that a conflict can happen between two cuts (e.g., c_1 and c_2 in Figure 5 (a)), two fills (e.g., f_2 and f_3 in Figure 5 (b)), or a fill on an inner shape and a cut on an outer shape (e.g., f_1 and c_2 in Figure 6 middle row).

Assuming that all candidates are nesting-aware, our optimization problem can now be stated simply as the following: *we seek a* $0/1$ *labelling* L *of the candidates* $\{X_1,\dots,X_n\}$ *that minimizes* $E(L)$ *defined in Equation 4 and is free of conflicts; that is, for any overlapping pair* $x\in X_i$ *and* $y\in X_{i+1}$ *and any* $i=1,\dots,n-1$, *either* $L(x)\neq 1$ *or* $L(y)\neq 0$.

5. Optimization

We explore several strategies to solve the conflict-free labelling problem formulated above, for a given set of nesting-aware candidates. We start with a heuristic approach that sequentially optimizes the labels in successive shapes while avoiding conflicts with previous shapes (Section 5.1). We then introduce a state-space search algorithm and prove that it always returns the optimal labelling, although with possibly exponential complexity (Section 5.2). Finally, we discuss a beam-search variant of the optimal algorithm that trades off optimality for efficiency (Section 5.3).

In all these strategies, we use the method of [ZCLJ20] as a black-box solver to optimize labels on individual shapes without considering conflicts. For convenience of discussion, we introduce two notations. Given two subsets F_0,F_1 of candidates X_i, we use $\mathcal{L}_i(F_0,F_1)$ to denote the labels of X_i computed by [ZCLJ20] to minimize $E(\{T_i^L\})$ (Equation 1) under the constraints that F_0 are all labelled 0 and F_1 are all labelled 1. Given subsets F_0,F_1 of all candidates on all shapes, we denote by $\mathcal{L}(F_0,F_1)$ the labelling of the entire sequence made up of per-shape labels $\mathcal{L}_i(F_0\cap X_i,F_1\cap X_i)$ for $i=1,\dots,n$.

5.1. Propagation

One idea is to "propagate" the labels from one shape to the remaining shapes in a sequential manner. During propagation, the labels of candidates X_i, denoted as L_i, are obtained by minimizing the labelling energy of L_i while constraining some labels to avoid conflicts with previously propagated labels in either L_{i-1} or L_{i+1}.

Specifically, starting from some $j\in[1,n]$, we first obtain the labels L_j of X_j as $\mathcal{L}_j(\emptyset,\emptyset)$ (i.e., without constraining any labels). The labels are then propagated forward and backward to adjacent shapes. In the forward direction, for each $i=j+1,\dots,n$, we define the constraint set $F_{i,1}$ as all candidates of X_i that overlap with some 1-labelled candidates in X_{i-1}:

$$F_{i,1}=\{x\in X_i | x\cap y\neq\emptyset,\exists y\in X_{i-1},L_{i-1}(y)=1\}$$

Candidates in this set must be labelled 1 to avoid conflicts with L_{i-1}. Hence we obtain $L_i=\mathcal{L}_i(\emptyset,F_{i,1})$. Similarly, in the backward direction, for each $i=j-1,\dots,1$, we define the constraint set $F_{i,0}$ as all candidates of X_i that overlap with some 0-labelled candidates in X_{i+1}:

$$F_{i,0}=\{x\in X_i | x\cap y\neq\emptyset,\exists y\in X_{i+1},L_{i+1}(y)=0\}$$

Since candidates in this set must be labelled 0 to avoid conflicts with L_{i+1}, we obtain $L_i=\mathcal{L}_i(F_{i,0},\emptyset)$. The forward and backward processes are illustrated in Figure 7 top and bottom.

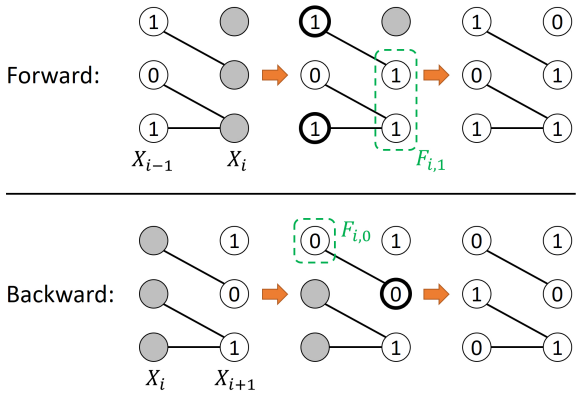

Figure 7: *Forward (top) and backward (bottom) propagation of labels. Candidates are shown as circles with their labels (gray circles have unknown labels) and overlapping candidates are connected by edges. Top: to get labels of candidates* X_i *from* X_{i-1}, *the subset* $F_{i,1}\subseteq X_i$ *that overlaps with 1-labelled candidates of* X_{i-1} *(thick outlines) is labelled 1, and the remaining labels are computed by energy minimization. Bottom: to get labels of candidates* X_i *from* X_{i+1}, *the subset* $F_{i,0}\subseteq X_i$ *that overlaps with 0-labelled candidates of* X_{i+1} *(thick outlines) is labelled 0, and the remaining labels are computed by energy minimization.*

Let L^j be the labelling over the entire sequence composed of the per-shape labels $\{L_1,\dots,L_n\}$ propagated from L_j. We perform propagation once from each starting index $j=1,\dots,n$, which results in n labellings $\{L^1,\dots,L^n\}$, and output the one with the least energy.

5.2. Optimal search

While the propagation method creates a conflict-free labelling, it is not guaranteed to be optimal in terms of its energy. We next

describe an *optimal* labelling algorithm (for the given set of candidates) based on state-space exploration. Here, a *state* is a triple $\{F_0, F_1, L\}$ where F_0 are candidates constrained to have label 0, F_1 are candidates constrained to have label 1, and $L = \mathcal{L}(F_0, F_1)$ is the energy-minimizing labelling subject to these constraints. The algorithm starts with a single state $\{\emptyset, \emptyset, \mathcal{L}(\emptyset, \emptyset)\}$, whose labelling minimizes the energy on each shape without enforcing nesting. At each iteration, the algorithm removes the state whose labelling achieves the least energy among all existing states. Let this state be $\{F_0, F_1, L\}$. If L is free of conflicts, the algorithm terminates and L is returned as the output. Otherwise, a conflict of L is chosen, and new states are created to resolve that conflict by adding more constraints to either F_0 or F_1.

The key step in the algorithm is the creation of new states. Consider a labelling L and a pair of overlapping candidates $x \in X_i$ and $y \in X_{i+1}$ with conflicting labels in L, that is, $L(x) = 1$ and $L(y) = 0$. A simple way to resolve the conflict is to constrain either x to have label 0 or y to have label 1. However, such constraints alone may lead to new conflicts with other shapes. For example, labelling x as 0 may conflict with 1-labelled candidates in X_{i-1} that overlap with x, and labelling y as 1 may conflict with 0-labelled candidates in X_{i+2} that overlap with y. To avoid such potential conflicts, our algorithm constrains not one, but possibly a set of candidates when resolving each conflict.

Specifically, we call two candidates $u \in X_i$ and $v \in X_j$ where $i < j$ *path-connected* if there is a sequence of candidates $\{w_i, \ldots, w_j\}$ such that $w_i = u$, $w_j = v$, $w_k \in X_k$ for $k = i, \ldots, j$, and each pair of consecutive candidates overlap (i.e., $w_k \cap w_{k+1} \neq \emptyset$ for $k = i, \ldots, j-1$). For each candidate $x \in X_i$, we define its 0-*set*, $\mathbf{0}(x)$, as the union of x and all candidates in X_j for $j < i$ that are path-connected with x, and its 1-*set*, $\mathbf{1}(x)$, as the union of x and all candidates in X_j for $j > i$ that are path-connected with x. To resolve the conflict involving a pair of candidates $\{x \in X_i, y \in X_{i+1}\}$, we constrain either all candidates in $\mathbf{0}(x)$ to have label 0 or all candidates in $\mathbf{1}(y)$ to have label 1. The constraints are appended to existing constraints in either F_0 or F_1, and the remaining labels are updated using energy minimization. For efficiency, we only update the labels of X_k, for $k = 1, \ldots, n$, if it has some constrained candidate z whose constraint label is different from its current label $L(z)$. This process is illustrated in Figure 8.

The search algorithm is summarized as pseudo-code in Algorithm 1. The algorithm maintains current states in a queue Q sorted by increasing labelling energy. Each iteration of the algorithm removes the state in Q with the least labelling energy and either returns the labelling, if it is conflict-free, or adds two new states to Q. We prove in Appendix A that the algorithm always terminates, and that the result is optimal when the solver $\mathcal{L}$ is optimal:

Proposition 5.1 Algorithm 1 terminates and produces a conflict-free labelling L in finite number of iterations. Furthermore, if $E(\mathcal{L}(F_0, F_1))$ is minimal among all labellings that label F_0 as 0 and F_1 as 1 for any two disjoint sets of candidates F_0, F_1, then $E(L)$ is minimal among all conflict-free labellings.

We make a final comment on the choice of the conflict candidate pair $\{x, y\}$ (Line 8 in Algorithm 1). Although the choice does not affect the optimality of the algorithm, it has an impact on the order of states being explored and therefore the overall running time. We

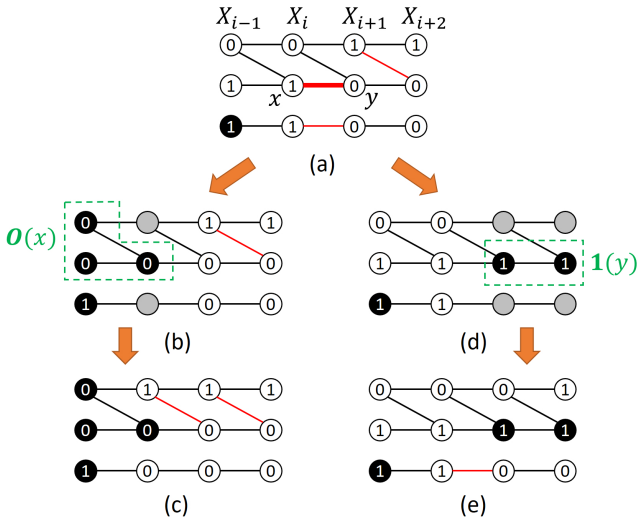

Figure 8: *State expansion. Given a state (a) consisting of candidate labels, constrained candidates (black circles), candidate pairs with conflicting labels (red edges), and a chosen pair $\{x, y\}$, two new states are created by expanding the constraints to include either the 0-set of x (b) or the 1-set of y (d) and updating the remaining labels on the affected shapes using energy minimization (c,e).*

Algorithm 1: State-space search algorithm

```
1  Initialize empty queue Q
2  Q.push({∅, ∅, L(∅, ∅)})
3  while Q is not empty do
4      {F_0, F_1, L} ⟵ Q.pop()
5      if L has no conflict then
6          return L
7      else
8          {x, y} ⟵ a conflict pair of candidates in L
9          F_0' ⟵ 0(x) ∪ F_0
10         F_1' ⟵ 1(y) ∪ F_1
11         Q.push({F_0', F_1, L(F_0', F_1)})
12         Q.push({F_0, F_1', L(F_0, F_1')})
13     end
14 end
```

adopt the following choice in our implementation. For each conflict $\{x, y\}$ of L, we count the number of 1-labelled candidates in $\mathbf{0}(x)$ and the number of 0-labelled candidates in $\mathbf{1}(x)$. We choose the conflict where the absolute difference between those two numbers is the smallest (ties are broken arbitrarily). This strategy prioritizes conflicts where one of the two ways of resolving it (either constraining $\mathbf{0}(x)$ to 0 or $\mathbf{1}(y)$ to 1) has a clearer benefit over the other in terms of minimizing the number of label-flippings in L.

5.3. Beam search

While being optimal, the search algorithm described above may have a prohibitive running time. As shown in Appendix A, the

states explored by the algorithm form a binary tree whose depth can be as large as the total number of candidates in all shapes. Denoting this number by m, the algorithm therefore may require $O(2^m)$ number of iterations. This can be impractical for inputs with many candidates, either due to a high level of topological noise or a large number of shapes in the sequence.

To make the algorithm more practical, at the cost of losing optimality, we can replace the full search by a beam search with a limited (and controllable) size of memory. Specifically, the maximum number of states that can be held in the queue Q is restricted to be a user-provided constant B, also known as the *beam width*. When Q is full and a new state needs to be pushed, the state whose labelling has the highest energy among the $B+1$ states is dropped from Q.

In contrast to the full search, the beam search requires only $O(Bm)$ number of iterations. In practice, we found that even a small beam width (e.g., 1) can produce near-optimal results but with significantly improved efficiency over the full search. In the extreme case that $B = 1$, Q holds only a single state at any time, and the algorithm simply replaces the current state at each iteration by one of the two newly created states that has a smaller labelling energy.

We illustrate the beam search with beam width $B = 1$ in a synthetic 2D example in Figure 9. This input consists of 3 shapes $\{T_1, T_2, T_3\}$, where T_1, T_2 each has multiple connected components and T_3 has a cavity. The labelling of the initial state (where no candidates are constrained) contains several conflicts. The algorithm picks one of the conflicting pairs of candidates, $\{f_1, f_2\}$, where the fill $f_1 \in X_1$ is labelled as 1 but the overlapping fill $f_2 \in X_2$ is labelled as 0. Among the two ways of resolving this conflict, the algorithm chooses to constrain the 0-set, $\mathbf{0}(f_1) = \{f_1\}$, to have label 0, and a new labelling is computed (iteration 1). The algorithm then picks another conflict, $\{f_3, c_4\}$, where the fill $f_3 \in X_2$ is labelled as 1 but the overlapping cut $c_4 \in X_3$ is labelled as 0. The conflict is resolved by constraining the 1-set, $\mathbf{1}(c_4) = \{c_4\}$, to have label 1, and a new labelling is computed (iteration 2). Since no conflict exists in the labelling, the algorithm terminates.

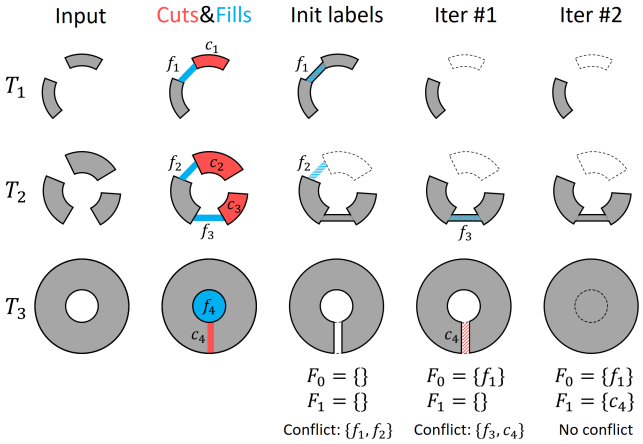

Figure 9: *Illustration of beam search on a 3-shape sequence with beam width $B = 1$. The columns (from left to right) show the input shapes, nesting-aware candidates, initial labelling of the candidates, and the labelling after 1st and 2nd iterations. The constrained candidate sets F_0, F_1 and the selected conflict to be resolved are noted at the bottom of each labelling.*

6. Results

We present experimental results that evaluate the effectiveness of our method, compare the different optimization strategies, and compare with alternative simplification methods.

While our method can be applied to any type of cell complexes and either way of shape definition from top-dimensional cells (see Figure 3), our current implementation is specialized to the common scenario of quadrilateral (in 2D) or cubical (in 3D) cell complexes, and it uses the shape definition of $\mathcal{C} \setminus \Omega(\mathcal{C}_d \setminus T_i)$ (Figure 3 (b)). This is equivalent to using 4-connectivity (in 2D) and 8-connectivity (in 3D) in digital topology [KR04]. In this case, preventing topological changes during deflation or inflation for computing nesting-aware candidates can be achieved by simply restricting the removal or addition operations to *simple* pixels and voxels [KR04].

Our code is implemented in C++, without parallelization, and uses the implementation provided by the authors of [ZCLJ20] for computing energy-minimizing candidate labels on individual shapes (we set their pruning parameter k to be 1). Our experiments are performed on a PC with an Intel(R) Core(TM) i9-10900X CPU (3.70GHz) and 64GB RAM.

6.1. Level sets

To compare different optimization options, we use the level sets of a grayscale 2D or 3D image at decreasing (or increasing) levels. Such "synthetic" sequences are always nested, and their complexity can be easily adjusted by changing the number of levels. Assuming the sequence consists of level sets with decreasing levels, we set the kernel K to be one of the pixels or voxels with the highest intensity (and hence is contained in the first shape) and the neighborhood N to be the entire image. Following [ZCLJ20], we set the geometric cost function $g_i(v)$ to be the magnitude of the intensity gradient of the image at a pixel or voxel v, which penalizes strong intensity edges, and prioritize pixels or voxels with higher (resp. lower) intensities during inflation (resp. deflation) to compute the nesting-aware candidates.

2D level sets (Figure 10): We first test on level sets extracted from a 2D retinal vessel image shown in (a). We took 6 expanding level sets from the image, as shown in Figure (b) with colors ranging from red to blue. Observe that these level sets have a large amount of topological noise including islands and cavities. We set up 5 experiments with increasingly longer input sequences, so that the i-th experiment uses the last $i+1$ level sets as the input. As seen in (e), the topological complexity of the input sequence, measured by the total Betti numbers $(\beta_0 + \beta_1)$ for all shapes in the sequence, increases as the experiment uses longer sequences.

In all five experiments, each of the three optimization approaches, namely propagation, optimal search, and beam search (with different beam width $B = 1, 2, 3$), is able to fully simplify the topology of the input sequence, meaning that each simplified

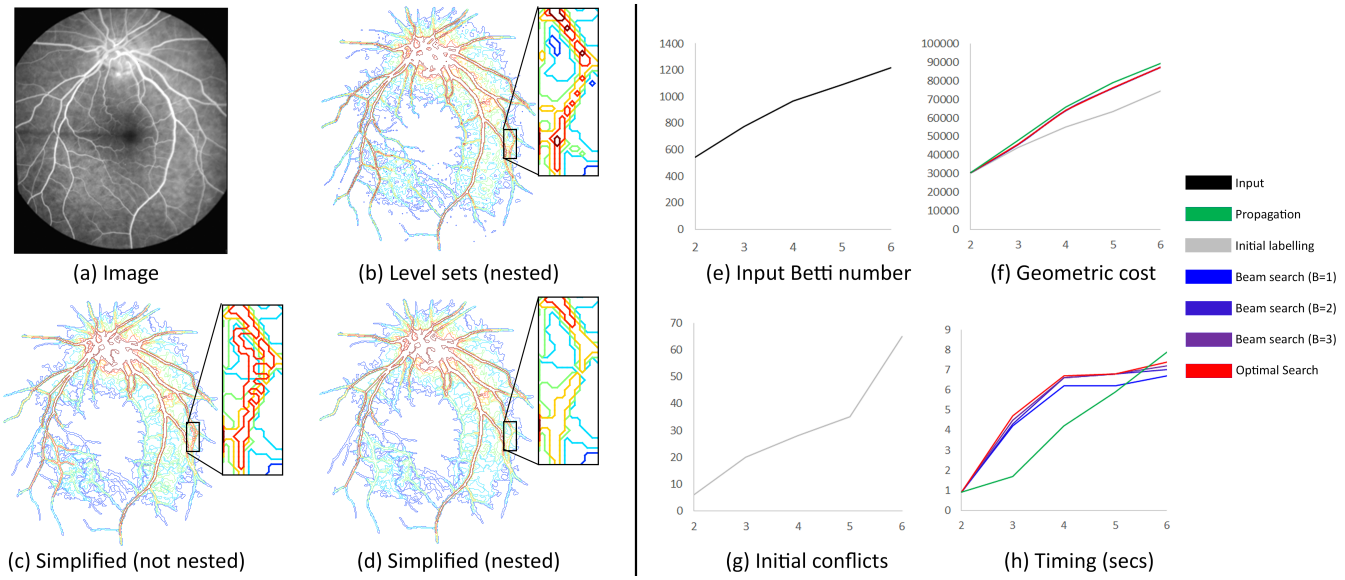

Figure 10: *Left: a grayscale retinal vessel image (a), 6 level sets used as input shapes (b), shapes simplified using the initial labelling (c) and conflict-free final labelling produced by our beam search ($B = 1$) (d). Right: plotting total Betti number of the input sequence (e), geometric costs of various optimization approaches (f), number of conflicts of the initial labelling (g), and running time of various optimization approaches (h) over experiments that use increasingly long input sequences.*

shape has a single component without cavities. We visually compare the initial labelling in the search algorithm, which is obtained by [ZCLJ20] without considering nesting, and the labelling returned by beam search with $B = 1$ in the last experiment (using all 6 level sets) in (c,d). While the initial labelling also results in topologically simple shapes, the simplified shapes could intersect (e.g., red, cyan and yellow curves in the close-ups of (c)), whereas the final labelling restores the nesting relation while maintaining topological simplicity. Beyond topology, different optimization approaches result in similar geometric costs, as plotted in (f), although propagation produces slightly higher costs. These costs become increasingly higher than the costs of the initial labelling, which correlates with the increase in the number of conflicts in the initial labelling as shown in (g).

In terms of performance, (h) shows that both optimal search and beam search have similar running time, while beam search with a smaller beam width is slightly faster. The propagation approach is generally faster than both search algorithms, although the advantage decreases with the length of the input sequence. This is because the propagation approach requires computing the energy-minimizing labels on all shapes for each starting shape, making its complexity quadratic to the number of shapes. In contrast, the beam search's complexity depends on a variety of factors such as the number of conflicts and the particular search path in the state space. Note that the time for computing the nesting-aware candidates is not included here, because they are common for all optimization approaches.

3D level sets (Figure 11): Our next test uses 7 level sets from a 3D CT image of a human foot bone, one of which is shown in (a). Observe that the bone shape has numerous handles, cavities, and islands in its interior. Similar as above, we set up 6 experiments using increasingly longer sequences of these level sets. The increase in topological complexity (as the total Betti number $\beta_0+\beta_1+\beta_2$ over all shapes) in these experiments is plotted in (e). To make the problem more challenging, we further removed some candidate cuts and fills so that the shapes cannot be fully simplified. Specifically, for each level set T_i, we removed all candidates from X_i that contain some voxel whose intensity is more than 40 away from the level of T_i (assuming the entire intensity range is 255).

Observe from (f) that both optimal search and beam search produce shapes with significantly fewer topological features than label propagation. Also, beam search (even with $B = 1$) produces only slightly more complex topology than optimal search. We visually compare in (b,c,d) the results of the propagation approach, initial labelling and final labelling of beam search for the input shape in (a). Observe that beam search results in fewer handles than propagation and better retains the appearance of the initial, energy-minimizing labelling. Performance-wise (h), propagation is notably faster than beam search, which is in turn significantly faster than optimal search, sometimes by two orders of magnitude. As a result, beam search (with $B = 1$) appears to offer the best compromise - it produces simplifications much closer to optimum than propagation but at a fraction of cost of the optimal search.

6.2. Real-world data

We applied our method to several real-world data sets including tissue layers in a brain scan and time series of growing plant roots. In each input sequence, all shapes are given as binary masks on a common voxel grid. We set the kernel K as one of the voxels inside the first shape that are furthest from the shape's boundary,

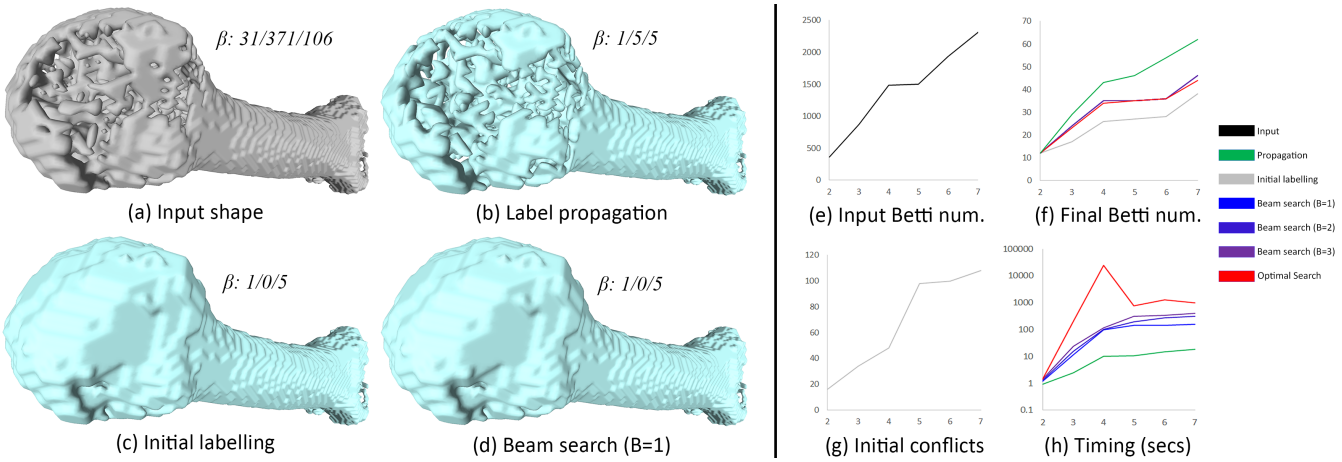

Figure 11: *Left: one of the seven level sets of a 3D CT image of a human foot bone that are used as input shapes (a), the shape simplified by label propagation (b), the initial labelling (c) and final conflict-free labelling of beam search ($B = 1$) (d). Betti numbers of each shape ($\beta_0/\beta_1/\beta_2$) are shown. Right: plotting total Betti numbers of the input sequence (e), total Betti numbers of the simplified sequence using various optimization approaches (f), number of conflicts of the initial labelling (g), and running time of various optimization approaches (h) over experiments that use increasingly long input sequences.*

and the neighborhood N as the entire grid. The geometric cost $g_i(v)$ is set to be constant for each voxel v, and the inflation and deflation prioritize voxels that are further away from the boundary of the shape being inflated or deflated towards.

We visually compare our method using the beam search strategy ($B = 1$) with the single-shape simplification method of [ZCLJ20] and the scalar-field simplification method of [TP12] implemented in the Topology Toolkit (TTK) [TFL*17]. For the latter, we first constructed a real-valued grayscale volume that interpolates the input shape sequence at integers equal to the indices of the shapes, then maximally simplified the extrema of the volume using TTK, and finally extracted the level sets of the simplified volume at the original integer values.

Brain layers (Figure 12): The input sequence (a) is constructed from 3 layers of a segmented human brain scan, namely (from inner to outer) the cerebrospinal fluid, white matter, and grey matter. The sequence consists of the inner most layer, the union of the inner and middle layers, and the union of all three layers. We refer to them as the inner, middle and outer shape, respectively. Numerous topological errors exist in this input, particularly on the inner and middle shapes (as seen in the close-ups), due to the complex anatomical structure. Both our method and [ZCLJ20] can remove all topological features on all three shapes, but the nesting relation between the shapes is only preserved in our method. In the result of [ZCLJ20] (b), the inner shape (red) can be seen outside the middle shape (gray) indicating a violation of nesting, while in our result (c) the inner shape is completely hidden inside the middle shape.

Growing roots (Figures 2,13): As roots grow in the soil or other media, and due to resistance of their surroundings, their shape become nested over time. Generally, roots are connected and free of handles or cavities, but the reconstruction of root shapes is prone to topological errors due to the presence of thin and nearby branches. Figure 2 compares our method with that of [ZCLJ20] on a sequence

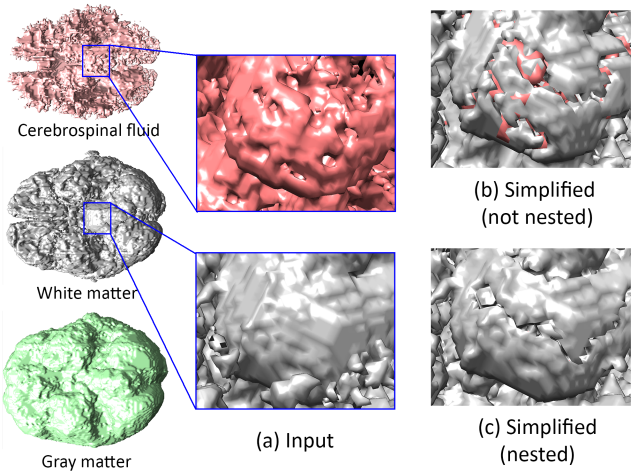

Figure 12: *(a) Three layers of a brain scan and a close-up on a region with complex topology in the two inner layers. (b) Two inner layers after applying the method of [ZCLJ20]; note the innermost layer extrudes outside the middle layer. (c) Our method keeps the innermost layer nested inside the middle layer.*

of 4 time points of a growing pennycress root system (showing only the last two time points). Again, while both methods fully simplify the topology, only our method preserves the nesting between the time points. Figure 13 shows another sequence of 3 time points of a growing rice root system and compares our method with TTK. As mentioned in Section 2.2, scalar-field simplification methods such as TTK guarantee the nesting relationship between the simplified level sets. However, these methods tend to have difficulties in removing topological handles on the level sets, which are caused by

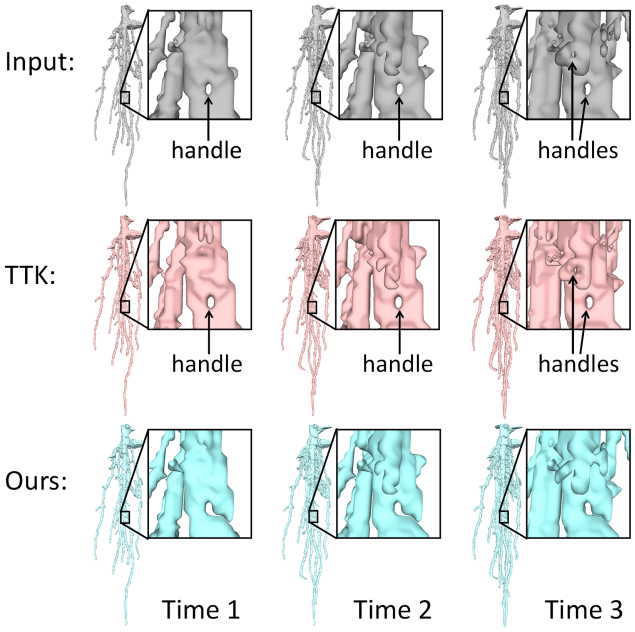

Figure 13: *Simplifying a sequence of 3 time points of a growing rice root system (top row) using TTK (middle row), which fails to remove most of the handles, and our method (bottom row) which fully simplifies the topology of each shape while maintaining nesting.*

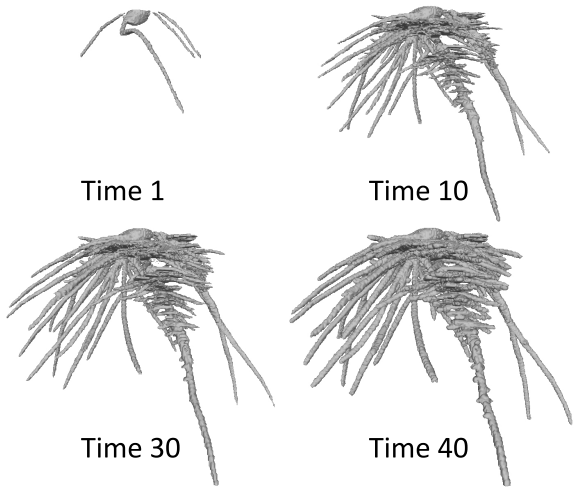

Figure 14: *Four time points of a growing maize root system from a sequence of 41 time points.*

saddle points in the scaler field.. Observe from the close-ups in Figure 13 (middle row) that TTK fails to remove several handles on the input root shapes. In contrast, our method removes all topological features (handles included) and keeps the shapes nested.

Quality and performance: Table 1 reports, for each real-world data set, the topological complexity of the input, running time for computing the nesting-aware candidates, and the cost (topology and geometry) and running time of the three optimization strategies ($B = 1$ for beam search). We additionally include a sequence of 41 time points of a growing maize root system (four time points are shown in Figure 14), which is our most complex data set. Observe that the optimal search is by far the most expensive of the three optimization approaches, and it failed to finish within ten hours for the maize data. Propagation is faster than beam search for the shorter sequences, but it takes significantly longer on the maize data due to the much longer sequence. Upon completion, all three strategies fully simplify the topology of these sequences, while optimal search and beam search have similar geometric costs that are lower than propagation. Once again, beam search seems to strike a better balance between optimality and speed.

7. Conclusions

We present, to the best of our knowledge, the first method for simultaneous topological simplification of a sequence of nested shapes while preserving nesting. Our method extends a recent global optimization method [ZCLJ20] to nested shapes by defining and computing nesting-aware candidate cuts and fills, formulating a constrained labelling problem using these candidates, and exploring several solution strategies that offer different trade-offs between optimality and efficiency. The method is shown to be effective on both synthetic and real-world shape sequences.

While our method produces topologically simple and nested shapes, the topological changes made by the algorithm may not be geometrically pleasing (e.g. the long "scar" on the surface of the white matter in Figure 12 (c)) and may destroy the semantics of the shape (e.g., the branching structure of the roots). The key to improving the quality of the results is obtaining candidate cuts and fills that better respect the geometry and semantics of the input shapes. We would also like to improve the efficiency of the current inflation/deflation procedure for computing the candidates, which currently accounts for a significant portion of the overall runtime. Furthermore, we will explore the extension of our current implementation to inputs represented on non-cubical grids (e.g., tetrahedral mesh or octrees). The primary change lies in the implementation of the inflation/deflation process, and in particular the decision on whether a cell can be removed or added without altering the topology of the shape. To make this process efficient, we will explore means to make such decisions using local information around a cell, similar to detecting simple voxels on cubical grids.

We see our work as the first step towards the general problem of *consistent* topological simplification of a collection of shapes. While our work considered a rather narrow interpretation of consistency (maintaining nesting in a shape sequence), we would like to explore other, and more general notions of consistency. A candidate notion is asking the surfaces of the topologically simplified shapes to have one-to-one correspondences with low geometric distortions. Such consistency would be useful in the context of simplifying shapes undergoing deformations (e.g., motion capture sequences) or similar shapes sharing a common topology (e.g., quadrupeds).

	Brain (3 layers)			Rice root (3 times)			Pennycress root (4 times)			Maize root (41 times)		
	$\beta_0/\beta_1/\beta_2$	Geom. cost	Time (sec)	$\beta_0/\beta_1/\beta_2$	Geom. cost	Time (sec)	$\beta_0/\beta_1/\beta_2$	Geom. cost	Time (sec)	$\beta_0/\beta_1/\beta_2$	Geom. cost	Time (sec)
Input	1247/863/80	-	-	352/133/4	-	-	408/254/0	-	-	12290/6780/201	-	-
Candidates	-	-	71.2	-	-	19.1	-	-	64.3	-	-	549.0
Propagation	3/0/0	897437.7	30.9	3/0/0	9931.2	1.3	4/0/0	3981.4	7.1	41/0/0	989392.5	26106.8
Opt. search	3/0/0	760882.1	248.6	3/0/0	9844.6	2.8	4/0/0	2765.3	43.1	N/A	N/A	N/A
Beam search	3/0/0	760882.1	71.4	3/0/0	9844.6	2.8	4/0/0	2773.3	9.1	41/0/0	709947.5	2664.1

Table 1: *Statistics for the four real-world data sets used in this paper (Figures 12, 13, 2, 14), showing the Betti numbers of the input sequence, time for computing the candidate cuts and fills, and the resulting topological complexity, geometric cost, and running time of the three optimization strategies.*

Acknowledgements

We appreciate the generous support from NSF (through grants DBI-1759836, EF-1921728, AF-1907612 and AF-2106672) and Washington University in St. Louis (through the Imaging Science Fellowship program). We would also like to thank Chris Topp at Donald Danforth Plant Science Center for providing the plant roots data set, and the anonymous reviewers for their suggestions.

References

[ABD*15] ATTALI D., BAUER U., DEVILLERS O., GLISSE M., LIEUTIER A.: Homological reconstruction and simplification in r3. *Computational Geometry 48*, 8 (2015), 606–621. 2, 4

[BHEP04] BREMER P.-T., HAMANN B., EDELSBRUNNER H., PASCUCCI V.: A topological hierarchy for functions on triangulated surfaces. *IEEE Transactions on Visualization and Computer Graphics 10*, 4 (2004), 385–396. 3

[BK02] BISCHOFF S., KOBBELT L.: Isosurface reconstruction with topology control. In *Pacific Conference on Computer Graphics and Applications* (2002), pp. 246–255. 2

[BLW12] BAUER U., LANGE C., WARDETZKY M.: Optimal topological simplification of discrete functions on surfaces. *Discrete & Computational Geometry 47*, 2 (2012), 347–377. 3

[CJL*18] CHAMBERS E. W., JU T., LETSCHER D., LI M., TOPP C.: Some heuristics for the homological simplification problem. In *CCCG* (2018), no. August. 2

[CW06] CHEN L., WAGENKNECHT G.: Automated topology correction for human brain segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention* (2006), Springer, pp. 316–323. 2

[ELZ02] EDELSBRUNNER H., LETSCHER D., ZOMORODIAN A.: Topological persistence and simplification. *Discrete & Computational Geometry 28*, 4 (2002). 3

[EMP06] EDELSBRUNNER H., MOROZOV D., PASCUCCI V.: Persistence-sensitive simplification functions on 2-manifolds. In *Proceedings of the twenty-second annual symposium on Computational geometry* (2006), pp. 127–134. 3

[GJR*14] GÜNTHER D., JACOBSON A., REININGHAUS J., SEIDEL H.-P., SORKINE-HORNUNG O., WEINKAUF T.: Fast and memory-efficienty topological denoising of 2d and 3d scalar fields. *IEEE transactions on visualization and computer graphics 20*, 12 (2014), 2585–2594. 3

[Hat02] HATCHER A.: *Algebraic Topology*. Cambridge University Pres, 2002. 3

[HLH*16] HEINE C., LEITTE H., HLAWITSCHKA M., IURICICH F., DE FLORIANI L., SCHEUERMANN G., HAGEN H., GARTH C.: A survey of topology-based methods in visualization. In *Computer Graphics Forum* (2016), vol. 35, Wiley Online Library, pp. 643–667. 3

[HXBNP02] HAN X., XU C., BRAGA-NETO U., PRINCE J. L.: Topology correction in brain cortex segmentation using a multi-scale, graph-based algorithm. *IEEE Trans. Med. Imaging 21*, 2 (2002), 109–121. 2

[JZH07] JU T., ZHOU Q.-Y., HU S.-M.: Editing the topology of 3d models by sketching. *ACM Trans. Graph. 26*, 3 (July 2007). 2

[KG01] KRIEGESKORTE N., GOEBEL R.: An efficient algorithm for topologically correct segmentation of the cortical sheet in anatomical mr volumes. *Neuroimage 14*, 2 (August 2001), 329–346. 2

[KR04] KLETTE R., ROSENFELD A.: *Digital geometry: Geometric methods for digital picture analysis*. Elsevier, 2004. 3, 8

[LGMT20] LUKASCZYK J., GARTH C., MACIEJEWSKI R., TIERNY J.: Localized topological simplification of scalar data. *IEEE Transactions on Visualization and Computer Graphics 27*, 2 (2020), 572–582. 3

[NT03] NOORUDDIN F. S., TURK G.: Simplification and repair of polygonal models using volumetric techniques. *IEEE Trans. Vis. Comput. Graph. 9*, 2 (2003), 191–205. 2

[PF09] PATANÈ G., FALCIDIENO B.: Computing smooth approximations of scalar functions with constraints. *Computers & Graphics 33*, 3 (2009), 399–413. 3

[SL01] SHATTUCK D. W., LEAHY R. M.: Automated graph based analysis and correction of cortical volume topology. *IEEE Trans. Med. Imaging 20*, 11 (2001), 1167–1177. 2

[SPF07] SÉGONNE F., PACHECO J., FISCHL B.: Geometrically accurate topology-correction of cortical surfaces using nonseparating loops. *IEEE transactions on medical imaging 26*, 4 (2007), 518–529. 2

[SV03] SZYMCZAK A., VANDERHYDE J.: Extraction of topologically simple isosurfaces from volume datasets. In *IEEE Visualization* (2003), pp. 67–74. 2

[TFL*17] TIERNY J., FAVELIER G., LEVINE J. A., GUEUNET C., MICHAUX M.: The topology toolkit. *IEEE transactions on visualization and computer graphics 24*, 1 (2017), 832–842. 10

[TP12] TIERNY J., PASCUCCI V.: Generalized topological simplification of scalar fields on surfaces. *IEEE transactions on visualization and computer graphics 18*, 12 (2012), 2005–2013. 3, 10

[WGS10] WEINKAUF T., GINGOLD Y., SORKINE O.: Topology-based smoothing of 2d scalar fields with c1-continuity. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 1221–1230. 3

[WHDS04] WOOD Z. J., HOPPE H., DESBRUN M., SCHRÖDER P.: Removing excess topology from isosurfaces. *ACM Trans. Graph. 23*, 2 (2004), 190–208. 2

[ZCLJ20] ZENG D., CHAMBERS E. W., LETSCHER D., JU T.: To cut or to fill: a global optimization approach to topological simplification. *ACM Trans. Graph. 39*, 6 (2020), 201–1. 2, 3, 4, 5, 6, 8, 9, 10, 11

[ZJH07] ZHOU Q.-Y., JU T., HU S.-M.: Topology repair of solid models using skeletons. *IEEE Transactions on Visualization and Computer Graphics 13*, 4 (July 2007), 675–685. 2

Appendix A: Proof of Proposition 5.1

We start with a few observations on the 0-sets and 1-sets. Recall that, for each candidate $x \in X_i$, its 0-set $\mathbf{0}(x)$ (resp. 1-set $\mathbf{1}(x)$) is the union of x and all candidates in X_j for $j < i$ (resp. $j > i$) that are path-connected with x.

We introduce two more definitions. Given a set of candidates S (possibly across multiple shapes), we say it is *0-inclusive* (resp. *1-inclusive*) if $\mathbf{0}(x) \subseteq S$ (resp. $\mathbf{1}(x) \subseteq S$) for any candidate $x \in S$. We show that:

Lemma A.1 The following statements hold for any candidates x, y and candidate sets S_1, S_2:

1. $x \in \mathbf{0}(y)$ if and only if $y \in \mathbf{1}(x)$.
2. $\mathbf{0}(x)$ is 0-inclusive and $\mathbf{1}(x)$ is 1-inclusive.
3. If S_1, S_2 are both 0-inclusive (resp. 1-inclusive), than $S_1 \cup S_2$ is also 0-inclusive (resp. 1-inclusive).

Proof We prove each statement in turn. We denote by $\pi(x)$ the index of the shape to which the candidate x belongs (i.e., $x \in X_{\pi(x)}$).

1. By definition of the 0-set, $x \in \mathbf{0}(y)$ if and only if x and y are path-connected and $\pi(x) < \pi(y)$. These are exactly the conditions for $y \in \mathbf{1}(x)$.
2. To show $\mathbf{0}(x)$ is 0-inclusive, we need to show that $\mathbf{0}(y) \subseteq \mathbf{0}(x)$ for any $y \in \mathbf{0}(x)$. Consider such a y and any $z \in \mathbf{0}(y)$. Since z is path-connected with y, which is in turn path-connected with x, z must be path-connected with x. Furthermore, $\pi(z) < \pi(y) < \pi(x)$. We conclude that $z \in \mathbf{0}(x)$, and hence $\mathbf{0}(y) \subseteq \mathbf{0}(x)$. A symmetric argument shows that $\mathbf{1}(x)$ is 1-inclusive.
3. We only need to show the 0-inclusive case; the 1-inclusive case is symmetric. For any $x \in S_1 \cup S_2$, x must either lie in S_1 or S_2. Suppose the former (the latter case is identical). Since S_1 is 0-inclusive, $\mathbf{0}(x) \subseteq S_1$ and hence $\mathbf{0}(x) \subseteq S_1 \cup S_2$, and therefore $S_1 \cup S_2$ is 0-inclusive as well.

□

Using these observations, we can show that each state created by the algorithm has two disjoint constraint sets that are respectively 0-inclusive and 1-inclusive:

Lemma A.2 The following statements hold for any state $\{F_0, F_1, L\}$ pushed into the queue Q in Algorithm 1:

1. $F_0 \cap F_1 = \emptyset$
2. F_0 is 0-inclusive and F_1 is 1-inclusive.

Proof We will prove by induction. Both (1,2) trivially hold for the initial state, where $F_0 = F_1 = \emptyset$. Consider a state $\{F_0, F_1, L\}$ popped by the algorithm. Assuming that (1,2) hold for F_0, F_1, we will show that F_0' obtained in Line 9 is (i) 0-inclusive and (ii) satisfying $F_0' \cap F_1 = \emptyset$. A symmetric argument would then show that F_1' obtained in Line 10 is 1-inclusive and satisfies $F_1' \cap F_0 = \emptyset$.

- (i): Since F_0 is 0-inclusive and so is $\mathbf{0}(x)$ (be Lemma A.2 (2)), their union F_0' is also inclusive (be Lemma A.2 (3)).
- (ii): Since $F_0 \cap F_1 = \emptyset$, we only need to show that $\mathbf{0}(x) \cap F_1 = \emptyset$. We first show that $x \notin F_1$. Suppose, on the contrary, that $x \in F_1$. Since $y \in \mathbf{1}(x)$ (because x, y overlap), and F_1 is 1-inclusive, we have $y \in F_1$, which contradicts with $L(y) = 0$. We next show that $\mathbf{0}(x) \cap F_1 = \emptyset$. Suppose on the contrary that there exists some $y \in \mathbf{0}(x)$ such that $y \in F_1$. By Lemma A.2 (1), $x \in \mathbf{1}(y)$, and hence $x \in F_1$, which contradicts to the earlier statement.

□

We are finally ready to prove Proposition 5.1:

Proof We first show termination. By Lemma A.1, the two constraint sets in each state are always disjoint, and hence a labelling under these constraints can always be found. Therefore the algorithm can always proceed. The states explored by the algorithm can be organized into a binary tree, where the popped state at each iteration becomes the parent of the two pushed states. Note that, since $F_0 \subset F_0'$ and $F_1 \subset F_1'$, the union of each child state's sets of constrained candidates is strictly larger than that of its parent. Since the total number of candidates is finite, the depth of the binary tree is finite, and so is the size of the tree. On the other hand, it is easy to see that the labelling associated with a state at the very bottom level of the tree, where all candidates are constrained, must be conflict-free (due inclusiveness of the constraints). Therefore the algorithm must terminate and return a conflict-free labelling (on Line 6) within finite number of iterations.

Next we show optimality. We call a state $\{F_0, F_1, L\}$ *compatible* with a labelling L' if L' labels all candidates in F_0 as 0 and all candidates in F_1 as 1. Note that, assuming optimality of $\mathcal{L}$, we always have $E(L) \leq E(L')$, because $L = \mathcal{L}(F_0, F_1)$ has the least energy among all labellings that satisfy the constraint sets F_0, F_1, and L' is one of such labellings.

Let L^* be a conflict-free labelling with minimal energy. We make a key observation that, at any time during the algorithm, there is at least one state in the queue Q that is compatible with L^*. We prove by induction. At the beginning of the algorithm, Q holds a single state where $F_0 = F_1 = \emptyset$, which is compatible with any labelling. As the algorithm proceeds, we need to show that, after a state $\{F_0, F_1, L\}$ compatible with L^* is popped from Q, at least one of the two pushed states remains compatible with L^*. To do so, note that since L^* is conflict-free, either $L^*(x) = 0$ or $L^*(y) = 1$ (both could hold). We first consider the case that $L^*(x) = 0$. By definition of the 0-set, L^* must assign label 0 to all candidates in $\mathbf{0}(x)$ to avoid conflicts. Since L^* already labels all of F_0 as 0 (due to compatibility), L^* therefore labels all of $F_0' = \mathbf{0}(x) \cup F_0$ as 0. This makes one of the pushed states, $\{F_0', F_1, \mathcal{L}(F_0', F_1)\}$, compatible with L^*. Similarly, if $L^*(y) = 1$, the other pushed state, $\{F_0, F_1', \mathcal{L}(F_0, F_1')\}$, is compatible with L^*.

To complete the proof, let L' be the labelling of the state in Q compatible with L^* when the algorithm terminates. Note that L' could be the same as the labelling L being returned. We have the following relation:

$$E(L) \leq E(L') \leq E(L^*)$$

The first inequality is due to the fact that L is the least-energy labelling in Q, and the second inequality is due to compatibility. By minimality of $E(L^*)$ and since L is also conflict-free, the equality must hold in this relation, which implies that L also has the minimal energy among all conflict-free labellings. □