

Fast, Simple and Separable Computation of Betti Numbers on Three-dimensional Cubical Complexes

Aldo Gonzalez-Lorenzo, Mateusz Juda, Alexandra Bac, Jean-Luc Mari, Pedro Real

▶ **To cite this version:**

Aldo Gonzalez-Lorenzo, Mateusz Juda, Alexandra Bac, Jean-Luc Mari, Pedro Real. Fast, Simple and Separable Computation of Betti Numbers on Three-dimensional Cubical Complexes. 6th International Workshop on Computational Topology in Image Context (CTIC 2016), Jun 2016, Marseille, France. pp.130-139, 10.1007/978-3-319-39441-1_12 . hal-01341014

HAL Id: hal-01341014
https://amu.hal.science/hal-01341014

Submitted on 23 Feb 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Fast, Simple and Separable Computation of Betti Numbers on Three-dimensional Cubical Complexes

Aldo Gonzalez-Lorenzo[1,2], Mateusz Juda[3] ⋆, Alexandra Bac[1], Jean-Luc Mari[1], and Pedro Real[2]

[1] Aix-Marseille Université, CNRS, LSIS UMR 7296, Marseille (France)
[2] University of Seville, Institute of Mathematics IMUS, Seville (Spain)
[3] Institute of Computer Science and Computational Mathematics, Jagiellonian University, Krakow (Poland)

Abstract. Betti numbers are topological invariants that count the number of holes of each dimension in a space. Cubical complexes are a class of CW complex whose cells are cubes of different dimensions such as points, segments, squares, cubes, etc. They are particularly useful for modeling structured data such as binary volumes.
We introduce a fast and simple method for computing the Betti numbers of a three-dimensional cubical complex that takes advantage on its regular structure, which is not possible with other types of CW complexes such as simplicial or polyhedral complexes. This algorithm is also restricted to three-dimensional spaces since it exploits the Euler-Poincaré formula and the Alexander duality in order to avoid any matrix manipulation. The method runs in linear time on a single core CPU. Moreover, the regular cubical structure allows us to obtain an efficient implementation for a multi-core architecture.

Keywords: Cubical complex, Betti numbers, 3D, separable, computational topology, homology

1 Introduction

Understanding a discrete volume can be addressed by determining its volume, its convexity, its diameter or any other geometrical descriptor. A higher level analysis can be made through topology, which tolerates continuous deformations. This could be seen as a less interesting approach, as we could not distinguish a sphere from a cube, but it actually furnishes a more essential information of the object. Homology is a powerful tool as its formalizes the concept of hole.

Holes of dimension 0, or 0-holes, correspond to connected components. 1-holes are tunnels or handles, which are particularly difficult to count in a volume depending on their shape. 2-holes correspond to voids in a volume. These notions

⋆ This research is supported by the Polish National Science Center under grant 2012/05/N/ST6/03621

can be generalized to higher dimensions, but they do not have an intuitive interpretation. We can compute the number of holes in each dimension or even draw them on the volume, though this is not useful with a complex shape.

Homology can be used for understanding an object without visualizing it, or to compare objects in a flexible way. It has been applied to dynamical systems [13,15], material science [4,18], electromagnetism [8,7], image understanding [1,14] and sensor networks [6].

In this article we aim at counting the number of holes (the Betti numbers) of a cubical complex embedded in a three-dimensional space. This is far from being an abstract work, as binary volumes (3D binary images, with voxels instead of pixels) can be transformed into equivalent cubical complexes. Our algorithm has a very specific input, since it cannot treat meshes or higher dimension cubical complexes, but it benefits from a good time complexity (linear) and a wide range of applications where data is structured in a lattice.

There have been a lot of works in computational homology in the last decades. Many of them [16,9,17] can compute the homology groups of more general spaces in cubical time. Computing only the Betti numbers (number of holes), which are the ranks of these groups, should be faster, but this has not been algorithmically proved. Delfinado and Edelsbrunner [5] introduce an algorithm with almost linear time complexity that computes the Betti numbers of a simplicial complex which is a subcomplex of a triangulation of S^3. The software library RedHom [12] is optimized for computing the homology in the context of cubical complexes. Wagner [19] also proposes an adapted algorithm for computing persistent homology on a cubical complex.

We propose an algorithm that is based on the computation of connected components and avoids any matrix manipulation. This is possible due to the Euler-Poincaré formula and the Alexander duality, which turn to be extraordinarily useful in the context of three-dimensional cubical complexes.

A simple description of the algorithm is given in Sect. 3. Then, we explain in Sect. 4 how to parallelize the computation by considering a different method for counting the connected components which is more adapted to the input data. Sections 5 and 6 explain the implementation of the algorithm and compare it with a previous software respectively.

2 Preliminaries

2.1 nD Cubical Complex

An *elementary interval* is an interval of the form $[k, k+1]$ or a degenerate interval $[k, k]$, where $k \in \mathbb{Z}$. An *elementary cube* is the Cartesian product of n elementary intervals, and the number of non-degenerate intervals in this product is its *dimension*. An elementary cube of dimension d will be called d-cube for short. Given two elementary cubes p and q, we say that p is a *face* of q if $p \subset q$.

The *Khalimsky coordinates* of an elementary cube $\prod_{i=1}^{n} [a_i, b_i]$ are $(a_1 + b_1, \cdots, a_n + b_n)$. The dimension of an elementary cube and its faces can be easily

deduced from its Khalimsky coordinates. For a cube q we denote its Khalimsky coordinates by $q[]$ and its ith component by $q[i]$.

An *nD cubical complex* is a set of elementary cubes. The *boundary* of a d-cube is the collection of its $(d-1)$-dimensional faces. By virtue of its regular structure, an nD cubical complex can be represented as an n-dimensional array (called CubeMap in [19]), where the cubes are represented by their Khalimsky coordinates.

From now on we assume that cubes of a given nD cubical complex K have all positive coordinates bounded by integers w_i $(1 \leq i \leq n)$. A_K is the binary n-dimensional array of size $L := \prod_{i=1}^{n}(2w_i + 1)$ where elementary cubes are represented by a Boolean equal to true associated to their Khalimsky coordinates. An element of the array with coordinates $x = (x_1, \ldots, x_n)$ is denoted by $A_K[x_1]\ldots[x_n]$ or $A[x]$ for short. The element $A_K[q[]]$ associated to the cube q is denoted by $A_K[q]$.

It is straightforward to provide an enumeration of Khalimsky coordinates in $\prod_{i=1}^{n}[0, 2w_i]$. Namely, there exists a bijection $I : \prod_{i=1}^{n}[0, 2w_i] \to [0, L-1]$. Such bijection I will be referred to as the *index map* and its image as the *index set*. For a cube q, $I(q)$ means $I(q[]) = I(q[1], \ldots, q[n])$.

The *support* of K, denoted by $\mathrm{supp}(K)$, is the nD cubical complex containing all the elementary cubes in $\prod_{i=1}^{n}[0, w_i]$. Thus, A_K encodes both K and $\mathrm{supp}(K)\backslash K$.

2.2 Homology

A *chain complex* (C, d) is a sequence of $\mathfrak{R}$-modules $C_0, C_1, \ldots$ (called *chain groups*) and homomorphisms $d_1 : C_1 \to C_0, d_2 : C_2 \to C_1, \ldots$ (called *differential* or *boundary operators*) such that $d_{q-1}d_q = 0$, for all $q > 0$, where $\mathfrak{R}$ is some ring, called the *ground ring* or *ring of coefficients*. In this paper we will fix $\mathfrak{R} = \mathbb{Z}_2$.

An nD cubical complex K induces a chain complex. C_q is the free $\mathfrak{R}$-module generated by the q-cubes of K. Its elements (called *q-chains*) are formal sums of q-cubes with coefficients in $\mathbb{Z}_2$, so they can be interpreted as sets of q-cubes. The linear operator d_q maps each q-cube to the sum of its $(q-1)$-dimensional faces.

A q-chain x is a *cycle* if $d_q(x) = 0$, and a *boundary* if $x = d_{q+1}(y)$ for some $(q+1)$-chain y. By the property $d_{q-1}d_q = 0$, every boundary is a cycle, but the reverse is not true: a cycle which is not a boundary contains a "hole". The qth homology group of the chain complex (C, d) contains the q-dimensional "holes": $H(C)_q = \ker(d_q)/\mathrm{im}(d_{q+1})$. This set is a finite-dimensional vector space, so there is a basis typically formed by the holes of the complex, whose elements are called *homology generators*. The ranks of the homology groups are called the *Betti numbers*, which count the number of holes in each dimension.

There is a slightly different homology theory called *reduced homology* where d_0 is defined otherwise. Thus, the zeroth Betti number β_0 is decremented by one. This avoids exceptional cases in several theorems.

3 The Algorithm

In this section we give a first presentation of our algorithm. It considers a restricted class of complexes: 3D cubical complexes. We explain in the following how we obtain each Betti number.

0th Betti number — It is well known that $\beta_0(K)$ is the number of connected components of K. This is easy to compute with a traversal of the complex.

2nd Betti number — Alexander duality relies the homology of a complex K of dimension 3 and its complementary in the three-dimensional sphere $S^3 \setminus K$.

Proposition 1 (Alexander duality) *Let K be a 3D cubical complex. Then $H_q(K)$ and $H^{2-q}(S^3 \setminus K)$ are isomorphic for reduced homology and cohomology.*

As a consequence, $\beta_2(K) = \beta_0(S^3 \setminus K) - 1$. That is, the number of voids in K is the number of connected components in the complementary minus one.

This result, which holds for more general spaces, is computationally interesting in the context of cubical complexes. First, the sphere S^n is easy to build. Figure 1 shows the spheres S^1 and S^2 as cubical complexes.

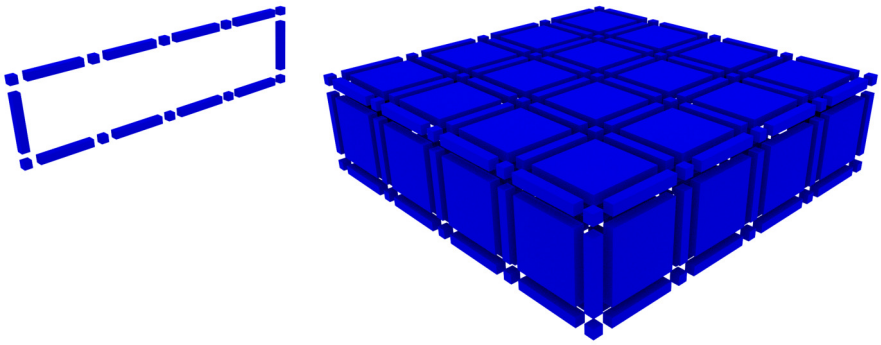


Fig. 1. Cubical complexes homeomorphic to S^1 and S^2.

Also, the complementary of a cubical complex is obvious to compute given its regular structure. Figure 2 illustrates the complementary of a cubical complex.

We want to obtain the number of connected components (minus one) of $S^3 \setminus K$ for deducing $\beta_2(K)$. Nevertheless, we do not need to build $S^3 \setminus K$. It suffices to count the connected components in $\text{supp}(K) \setminus K$ and consider only those which do not contain a cube in the boundary of $\text{supp}(K)$. These connected components are connected to $S^3 \setminus \text{supp}(K)$, thus making only one connected component in $S^3 \setminus K$. Note that this fact is far easier to understand for a 1D or a 2D cubical complex.

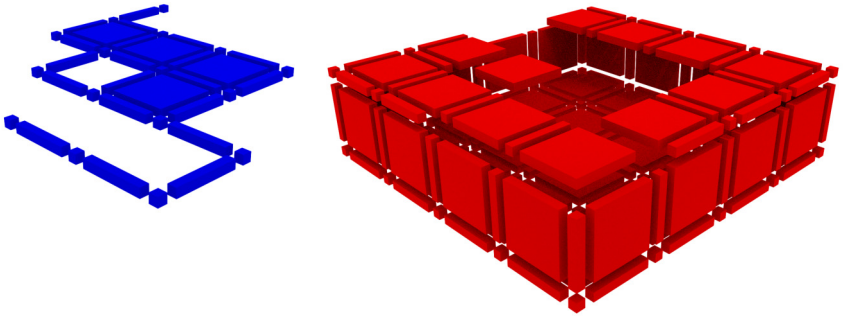


Fig. 2. A two-dimensional cubical complex K and its complementary $S^2 \setminus K$

1st Betti number — Once $\beta_0(K)$ and $\beta_2(K)$ are known, $\beta_1(K)$ is easy to obtain via the Euler-Poincaré formula. The *Euler-Poincaré characteristic* of a 3D cubical complex K is the alternating sum of its cubes. Formally,

$$\chi(K) = k_0 - k_1 + k_2 - k_3,$$

where k_q denotes the number of cubes of dimension q in K. This number, which is easy to compute, is a topological invariant.

Proposition 2 (Euler-Poincaré formula) *Let K be a 3D cubical complex. Then $\chi(K) = \beta_0(K) - \beta_1(K) + \beta_2(K)$.*

Therefore, $\beta_1(K) = \beta_0(K) + \beta_2(K) - \chi(K)$.

Algorithm 1 combines these three ideas. It passes by all the elements of A_K and traverses the connected components of K and $\text{supp}(K) \setminus K$. For the sake of simplicity we do not explicitly describe the computation of $\chi(K)$ in Algorithm 1. It can be obtained by adding $\chi \leftarrow \chi + (-1)^{\dim(p)}$ to line 16. As each cube is connected to six other cubes in A_K (except for the cubes in the boundary of A_K), the complexity of the algorithm is $O(n+6n) = O(n)$ where n is the number of cubes in $\text{supp}(K)$.

4 Recursive version of the Algorithm

The core of the previous algorithm is the computation of connected components through a traversal of the three-dimensional array A_K. This is difficult to parallelize because it uses a queue data structure. In this section we describe an algorithm for computing connected components of an nD cubical complex K in parallel. The algorithm total CPU utilization (i.e. work) is almost linear. It significantly uses the representation of a cubical complex as a multidimensional array A_K with an index map I.

Algorithm 1 BettiViaCC

Input: K a 3D cubical complex; A_K its associated binary array
Output: The Betti numbers of K: β_0, β_1, β_2

$\beta_0 \leftarrow 0$, $\beta_2 \leftarrow 0$
for all $p \in A_K$ not marked **do**
  $b \leftarrow$ false
  $Q \leftarrow$ an empty queue
  $Q.push(p)$; mark p
  while Q not empty **do**
    $q \leftarrow Q$.pop()
    if q belongs to the boundary of A_K **then**
      $b \leftarrow$ true
    end if
    for all q' 6-neighbor of q, $A_K[q'] = A_K[q]$, q' not marked **do**
      Q.push(q'); mark q'
    end for
  end while
  if $A_K[p] =$ true **then**
    $\beta_0 \leftarrow \beta_0 + 1$
  else if $b =$ false **then**
    $\beta_2 \leftarrow \beta_2 + 1$
  end if
end for
$\beta_1 \leftarrow \beta_0 + \beta_2 - \chi(K)$
return $(\beta_0, \beta_1, \beta_2)$

In Sect. 3 we count connected components by traversing the connectivity graph of the cubical complex. Another well known approach to compute connected components is to use disjoint set data structure. The data structure maintains a collection $S = \{ S_1, \ldots, S_k \}$ of disjoint sets. Each set in S is identified by a representative, which is a member of the set (see [3, Chapter 21]). The following operations may be performed on the disjoint set data structure C:

- $C.\,\mathrm{makeSet}(x)$ - creates a new set whose only member (and thus representative) is x.
- $C.\,\mathrm{find}(x)$ - returns a pointer to the representative of the (unique) set containing x.
- $C.\,\mathrm{union}(x, y)$ - merges the sets that contain x and y into a new set that is the union of these two sets.

To compute connected components of a cubical complex it is enough to call $C.\,\mathrm{union}(x, y)$ for each pair x, y of adjacent cubes. A parallel version of such algorithm requires synchronization, so in practice it cannot be implemented efficiently. However, the regular structure of a cubical complex allows us to propose a different approach where synchronization is not needed. The idea is to recursively cut the complex in two halves, find the connected components in each half and then merge them.

Let K be a cubical complex and I the index map of Khalimsky coordinates. Let J be a subset of the index set associated with K. We define $K_J := \{ q \in K \mid I(q) \in J \}$. We also define the *left slice*, *right slice* and *middle slice* of J in dimension d by x respectively as

$$\begin{aligned} S(J, x_-, d) &:= \{ y \in J \mid I^{-1}(y)[d] < x \} \\ S(J, x^+, d) &:= \{ y \in J \mid x \leq I^{-1}(y)[d] \} \\ S(J, x, d) &:= \{ y \in J \mid x - 1 \leq I^{-1}(y)[d] \leq x \}. \end{aligned}$$

For a $j \in J$ we denote by $\mathrm{cc}_J(j)$ the connected component of K_J to which j belongs. Algorithm 2 computes recursively connected components of a cubical complex. Observe that at each step of the recursion the set J is split following some rule. We do not give an explicit description of the rule, but it should divide J into two sets of similar size by separating K_J along alternate axes. We thus obtain three subsets that cover J, one of them intersecting the other two so we can merge the connected components computed on each side. The first two recursive steps (lines 4 and 5) work on independent data, so they can be executed in parallel. The third recursive step at line 6 always jumps to the line 8 (since $J \not> \epsilon = \infty$) and it depends on the previous two steps.

Algorithm 2 RecursiveCC

Input: K a 3D cubical complex; I its associated index map; $J \subset I$.
Input: C a disjoint set data structure on the index set of K, such that $C.\mathrm{find}(i) \neq C.\mathrm{find}(j)$ for all $i, j \in J$.
Input: Parameters: $d \in \mathbb{Z}$ and $\epsilon > 0$.
Output: For each pair $i, j \in J$ we have $\mathrm{cc}_J(i) = \mathrm{cc}_J(j)$ if and only if $C.\mathrm{find}(i) = C.\mathrm{find}(j)$.

```
 1: if size of J > ε then
 2:     d ← using d choose dimension for next slicing
 3:     x ← choose slicing value in dimension d
 4:     RecursiveCC(K, I, S(J, x_−, d), d, ε, C)
 5:     RecursiveCC(K, I, S(J, x^+, d), d, ε, C)
 6:     RecursiveCC(K, I, S(J, x, d), d, ∞, C)
 7: else
 8:     for all p ∈ K_J do
 9:         for all q 2n-neighbor of p in K_J do
10:             C.union(I(p), I(q))
11:         end for
12:     end for
13: end if
```

Algorithm 3 computes the Betti numbers of a 3D cubical complex K. It computes the connected components of K and $\mathrm{supp}(K) \setminus K$ in two calls to Algorithm 2. Again, $\chi(K)$ can be computed during the traversal of the complex.

Algorithm 3 RecursiveBetti

Input: K a 3D cubical complex; I its associated index map.
Input: Parameter $\epsilon > 0$.
Output: The Betti numbers of K: β_0, β_1, β_2.

$C_1 \leftarrow$ a disjoint set for $\operatorname{im} I$
for all $q \in K$ **do**
    $C_1.\operatorname{makeSet}(I(q))$
end for
$\operatorname{RecursiveCC}(K, I, \operatorname{im} I, 0, \epsilon, C_1)$
$\beta_0 \leftarrow$ number of sets in C_1

$K_0 \leftarrow \operatorname{supp}(K) \setminus K$
$C_0 \leftarrow$ a disjoint set for $\operatorname{im} I$
for all $q \in K_0$ **do**
    $C_0.\operatorname{makeSet}(I(q))$
end for
$\operatorname{RecursiveCC}(K_0, I, \operatorname{im} I, 0, \epsilon, C_0)$
$r \leftarrow$ number of sets in C_0 containing a cube in the boundary of $\operatorname{supp}(K)$
$\beta_2 \leftarrow$ number of sets in C_0 minus r
$\beta_1 \leftarrow \beta_0 + \beta_2 - \chi(K)$
return $(\beta_0, \beta_1, \beta_2)$

5 Implementation

Algorithm 3 is implemented as a part of the CAPD::RedHom project [11]. Our parallel version of the implementation uses Threading Building Blocks library [10]. A crucial part of the implementation is a data structure for efficient slicing of the index set. For this we use Boost.MultiArray, a library from Boost Project [2]. It is an implementation of a multidimensional array container. In our case the data structure contains the index set. It provides an efficient slicing operation implemented as views to the original container. We use it to implement the operation S from the algorithm. At each recursion step we take a direction an cut the multidimensional array in the middle of the direction.

The data structure provides a mapping from multidimensional indices (in our case Khalimsky coordinates) to the index set. Technically it is enough to implement a mapping from the set of indices to a linear space of memory $[0, L-1]$ containing the value i at the ith position. Taking advantage of this fact, features of the C++ language, and Boost.MultiArray, we do not have to allocate memory for the index set. We get the index set and the slicing operation without any additional cost. Of course we can achieve it in many ways, however with our approach we can reuse well tested code.

6 Validation

Table 1 shows results of numerical experiments with the algorithm implementation. We compare also with standard approach for Betti numbers computa-

tions using elementary reductions, coreduction, and Morse decomposition from CAPD::RedHom [11]. All the computations were performed using one data structure, only algorithms vary.

Data sets N0001 and P0001 come from computer assisted proofs in dynamics. Data sets rand_pP_S were generated randomly, where S is the size of the grid and each 3-cube (together with its faces) is included with probability P. The data sets are in binary format, thus reading time can be omitted. Computations were performed on a 2,3 GHz Intel Core i7 (4 real cores, 8 virtual) with 16GB RAM. The results show that the parallel implementation is around 4 times faster than the sequential one. It suggest a perfect scalability with the number of real cores. Also, we see that for the new algorithm only grid size matters.

Table 1. CPU time (format [h:]mm:ss) usage for cubical complexes. Computations with following algorithms from CAPD::RedHom: Alg. 3 parallel, Alg. 3 sequential, standard

			Parallel	Sequential	Standard
Data set	Grid size	Number of cells	CPU	CPU	CPU
N0001	256^3	75357994	0:23	1:18	1:31
P0001	256^3	75559573	0:23	1:18	1:39
rand_p25_256	256^3	75897341	0:22	1:13	3:35:22
rand_p50_256	256^3	110450571	0:23	1:15	
rand_p75_256	256^3	127326478	0:23	1:17	
rand_p25_384	384^3	256006045	1:21	4:12	> 4h
rand_p50_384	384^3	372383238	1:18	4:17	
rand_p75_384	384^3	429007477	1:17	4:15	

7 Conclusion

This paper introduces a linear algorithm that computes the Betti numbers of a 3D cubical complex. It counts the connected components of the complex and its complementary in S^3 and uses the Euler-Poincaré formula. The algorithm is specially conceived for cubical complex as it takes advantage of its regular structure both in a theoretical and a practical manner. It cannot be extended to 4D cubical complexes since the Euler-Poincaré formula does not suffices to obtain all the Betti numbers.

An interesting issue that should be addressed in the near future is how to adapt this algorithm for simplicial complexes. The main problem is that we need a triangulation of the complementary of the complex in S^3, which is not as easy as for cubical complexes.

The current implementation outperforms the existing software for computing Betti numbers on cubical complexes. It is available as a part of the CAPD::RedHom [11] project. A more detailed comparison will be done in a forthcoming paper.

References

1. M. Allili and D. Corriveau. Topological analysis of shapes using Morse theory. *Computer Vision and Image Understanding*, 105(3):188–199, 2007.
2. BoostCommunity. Boost Project. http://www.boost.org/, 2016.
3. Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.
4. Sarah Day, William D. Kalies, and Thomas Wanner. Verified homology computations for nodal domains. *Multiscale Modeling & Simulation*, 7(4):1695–1726, 2009.
5. Cecil Jose A. Delfinado and Herbert Edelsbrunner. An incremental algorithm for Betti numbers of simplicial complexes on the 3-sphere. *Computer Aided Geometric Design*, 12(7):771–784, 1995.
6. Pawel Dlotko, Robert Ghrist, Mateusz Juda, and Marian Mrozek. Distributed computation of coverage in sensor networks by homological methods. *Applicable Algebra in Engineering, Communication and Computing*, 23(1-2):29–58, 2012.
7. Paweł Dłotko and Ruben Specogna. Efficient cohomology computation for electromagnetic modeling. *CMES: Computer Modeling in Engineering & Sciences*, 60(3):247–278, 2010.
8. Paul W. Gross and P. Robert Kotiuga. *Electromagnetic Theory and Computation*. Cambridge University Press, 2004. Cambridge Books Online.
9. Shaun Harker, Konstantin Mischaikow, Marian Mrozek, and Vidit Nanda. Discrete morse theoretic algorithms for computing homology of complexes and maps. *Foundations of Computational Mathematics*, 14(1):151–184, 2013.
10. Intel. Threading Building Blocks. https://www.threadingbuildingblocks.org/, 2016.
11. M. Juda, M. Mrozek, P. Brendel, H. Wagner, and et al. CAPD::RedHom. http://redhom.ii.uj.edu.pl, 2010-2016.
12. Mateusz Juda and Marian Mrozek. CAPD::RedHom v2 - homology software based on reduction algorithms. In Hoon Hong and Chee Yap, editors, *Mathematical Software - ICMS 2014*, volume 8592 of *Lecture Notes in Computer Science*, pages 160–166. Springer Berlin Heidelberg, 2014.
13. Konstantin Mischaikow. Conley index theory. In Russell Johnson, editor, *Dynamical Systems*, volume 1609 of *Lecture Notes in Mathematics*, pages 119–207. Springer Berlin Heidelberg, 1995.
14. M. Mrozek, M. Zelawski, A. Gryglewski, S. Han, and A. Krajniak. Homological methods for extraction and analysis of linear features in multidimensional images. *Pattern Recognition*, 45(1):285–298, 2012.
15. Marian Mrozek. Index pairs algorithms. *Foundations of Computational Mathematics*, 6:457–493, 2006.
16. James R. Munkres. *Elements of algebraic topology*. Addison-Wesley, 1984.
17. Samuel Peltier, Sylvie Alayrangues, Laurent Fuchs, and Jacques-Olivier Lachaud. Computation of homology groups and generators. In Eric Andres, Guillaume Damiand, and Pascal Lienhardt, editors, *Discrete Geometry for Computer Imagery*, volume 3429 of *Lecture Notes in Computer Science*, pages 195–205. Springer Berlin Heidelberg, 2005.
18. Takashi Teramoto and Yasumasa Nishiura. Morphological characterization of the diblock copolymer problem with topological computation. *Japan Journal of Industrial and Applied Mathematics*, 27(2):175–190, 2010.
19. Hubert Wagner, Chao Chen, and Erald Vuçini. Efficient computation of persistent homology for cubical data. In Ronald Peikert, Helwig Hauser, Hamish Carr, and

Raphael Fuchs, editors, *Topological Methods in Data Analysis and Visualization II*, Mathematics and Visualization, pages 91–106. Springer Berlin Heidelberg, 2012.