Computational topology.

Paweł Dłotko

13/03/2013 IAS.

General scheme of homology computations on a computer.

Set

General scheme of homology computations on a computer.

Set

Topological space

General scheme of homology computations on a computer.

Set

Topological space

Nodal domain of a function

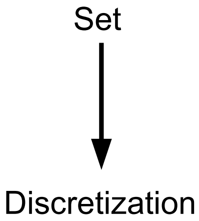
General scheme of homology computations on a computer.

$$\begin{tikzcd}\text{Set} \arrow[d] \\ \text{Discretization}\end{tikzcd}$$

Topological space

Nodal domain of a function

General scheme of homology computations on a computer.



Topological space

Nodal domain of a function

Simplicial complex

General scheme of homology computations on a computer.

$$\begin{tikzcd}\text{Set} \arrow[d] \\ \text{Discretization}\end{tikzcd}$$

Topological space

Nodal domain of a function

Simplicial complex

Cubical complex

General scheme of homology computations on a computer.

$$\begin{tikzcd}\text{Set} \arrow[d] \\ \text{Discretization}\end{tikzcd}$$

Topological space

Nodal domain of a function

Simplicial complex

Cubical complex

Regular CW complex

General scheme of homology computations on a computer.

$$\begin{tikzcd}
\text{Set} \arrow[d] \\
\text{Discretization} \arrow[d] \\
\text{Topological information}
\end{tikzcd}$$

Topological space

Nodal domain of a function

Simplicial complex

Cubical complex

Regular CW complex

General scheme of homology computations on a computer.

$$\begin{tikzcd} \text{Set} \arrow[d] \\ \text{Discretization} \arrow[d] \\ \text{Topological information} \end{tikzcd}$$

Topological space

Nodal domain of a function

Simplicial complex

Cubical complex

Regular CW complex

Betti numbers, torsion coefficients

General scheme of homology computations on a computer.

$$\begin{tikzcd}
\text{Set} \arrow[d] \\
\text{Discretization} \arrow[d] \\
\text{Topological information}
\end{tikzcd}$$

Topological space

Nodal domain of a function

Simplicial complex

Cubical complex

Regular CW complex

Betti numbers, torsion coefficients

Persistent diagrams

General scheme of homology computations on a computer.

$$\begin{tikzcd}
\text{Set} \arrow[d] \\
\text{Discretization} \arrow[d] \\
\text{Topological information}
\end{tikzcd}$$

Topological space

Nodal domain of a function

Simplicial complex

Cubical complex

Regular CW complex

Betti numbers, torsion coefficients

Persistent diagrams

(co)homolgy generators

General scheme of homology computations on a computer.

$$\begin{tikzcd}
\text{Set} \arrow[d] \\
\text{Discretization} \arrow[d] \\
\text{Topological information}
\end{tikzcd}$$

Topological space

Nodal domain of a function

Simplicial complex

Cubical complex

Regular CW complex

Betti numbers, torsion coefficients

Persistent diagrams

(co)homolgy generators

Representation of spaces.

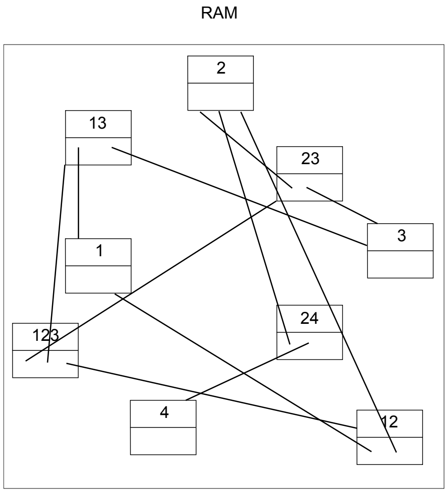
1. (Abstract) simplicial complex.
2. Cubical complex.
3. Regular CW-complex.
4. Chain complex.
5. For reduction methods we need to iterate among neighboring cells fast.

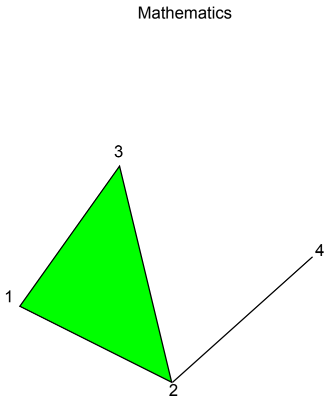
Abstract simplicial complex.

```
class Simplex
{
//..
vector<Simplex*> faces;
vector<Simplex*> cofaces;
double filtrationLevel;
int id;
//some extra information
};

class AbstractSimplicialComplex
{
//..
vector< vector<Simplex*> > elements;
//graded vector of simplices;
};
```

Representation in RAM vs Mathematics.

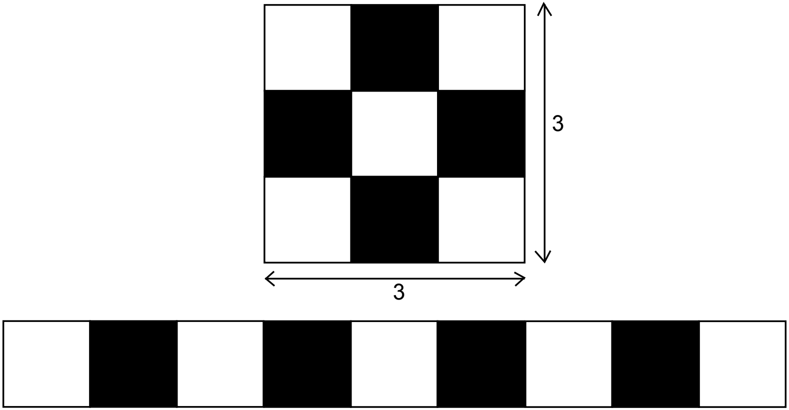




Regular vs not regular grids.

1. Simplicial complexes do not have regular structure.
2. The size of data structure to store simplex $\sim$ number of its faces times 32 bits.
3. Regular grids allows us to decrease drastically amount of memory needed to store the structure.
4. Possible to get neighbors of a given cell from the structure of a grid.
5. Cubical grids with cubes having equal sizes forms regular grids.
6. 1 bit of information per cube suffices (for complex) or 1 real number (8 bytes $=$ 64 bits) for complex with filtration.

The Bitmap (basic version).

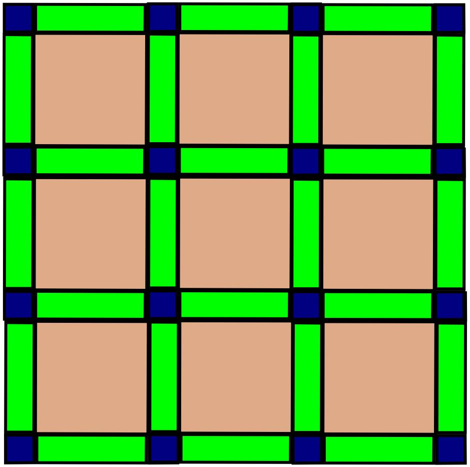


bool* bitmap; //!! Be careful!
char* bitmap; //This is the right way in C++.

Regular vs not regular grids.

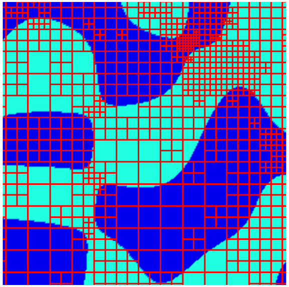
1. One can build bitmap in any dimension, 2d is just to present the idea.
2. All the bitmaps caches very well.
3. Only top dimensional cubes are represented.
4. Elementary reductions or acyclic subspace method can be used.
5. After reduction, the chain complex of the remaining set is created.
6. There are bitmaps in which all the cells of complex are represented.

The Bitmap.



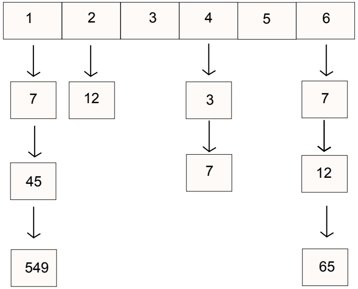
Rectangular / Regular CW-complex.

1. Obtained as topologically faithful representation of a nodal domain of a function.
2. They are represented in the same way as simplicial complexes (vectors of pointers).
3. In case of nodal domains of random trig. polynomials this is a better representation then a bitmap of uniform size cubes.



Sparse matrix representation (for chain complexes over $\mathbb{Z}_2$).

1. General chain complex is stored as a sparse matrix.
2. They are usually represented as a vector of lists.
3. For easy access to coboundary it can be stored by using the same structure.



Other representations.

1. Rectangular CW-complexes (quad-trees, oct-trees).
2. Point clouds - Rips complexes via Rips graph.
3. Voronoi diagrams.
4. Some representation for 2-dimensional simplicial complexes used in computer graphics.

What is the best representation for your purposes?

1. Everything here is just my intuition.
2. No universal data structure – the best one depends on your data and problem to solve.
3. For regular cubical set, bitmap representation is natural.
4. If the cubes have different sizes, representation of rectangular set may be better.
5. For simplicial complex if access to (co)boundary is needed – try pointer representation.
6. Almost always pays off to use sparse matrix data structure unless your data are very small.

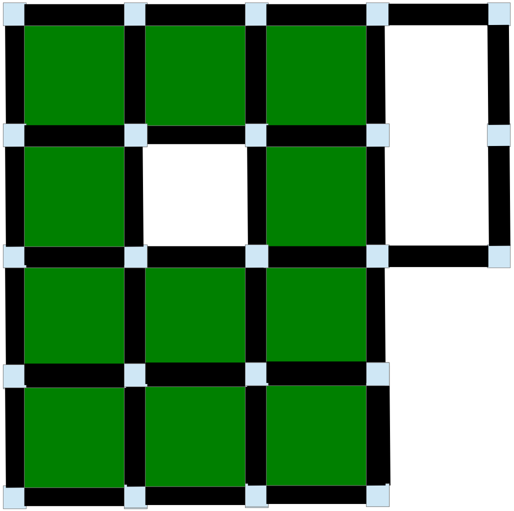
Reduction methods.

1. Reduction methods were developed as heuristic to enable homology computations for big data.
2. Elementary reduction aka. free face collapsing.
3. Coreductions.
4. Acyclic subspace method.
5. Algebraic reductions (KMS).
6. I will give some intuitive demonstration why they work based on Discrete Morse Theory.

Procedure 1 Elementary reduction.

```
Input: vector of cell complex;
  queue T;
  for every a in complex do
     if a has unique coboundary element then
        T.enqueue( a );
  while T is not empty do
     cell a = T.dequeue();
     if a has unique element b in coboundary then
        remove a and b from complex;
        for every c face of b such that c!=a do
           if c has unique coboundary element then
              T.enqueue(c);
        for every c face of a do
           if c has unique coboundary element then
              T.enqueue(c);
```

Elementary Reductions.

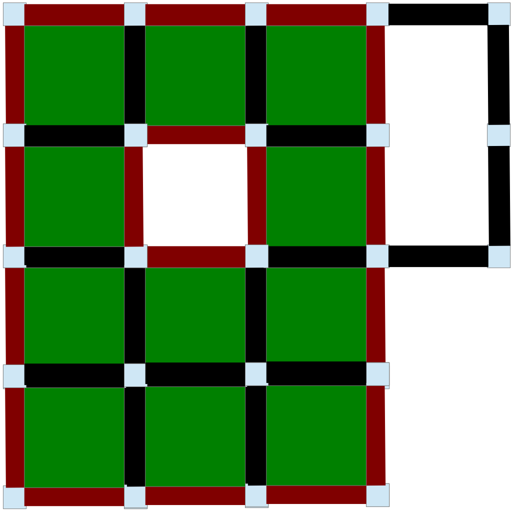


Elementary reductions

Procedure 2 Elementary reduction.

```
Input: vector of cell complex;
  queue T;
  for every a in complex do
    if a has unique coboundary element then
      T.enqueue( a );
  while T is not empty do
    cell a = T.dequeue();
    if a has unique element b in coboundary then
      remove a and b from complex;
      for every c face of b such that c!=a do
        if c has unique coboundary element then
          T.enqueue(c);
      for every c face of a do
        if c has unique coboundary element then
          T.enqueue(c);
```

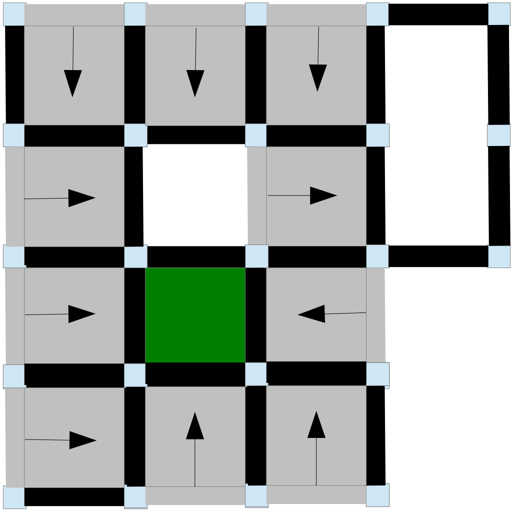
Elementary Reductions.



Procedure 3 Elementary reduction.

```
Input: vector of cell complex;
  queue T;
  for every a in complex do
    if a has unique coboundary element then
      T.enqueue( a );
  while T is not empty do
    cell a = T.dequeue();
    if a has unique element b in coboundary then
      remove a and b from complex;
      for every c face of b such that c!=a do
        if c has unique coboundary element then
          T.enqueue(c);
      for every c face of a do
        if c has unique coboundary element then
          T.enqueue(c);
```

Elementary Reductions.

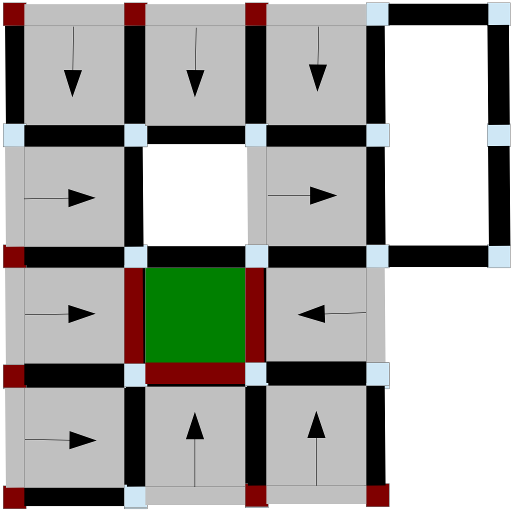


Procedure 4 Elementary reduction.

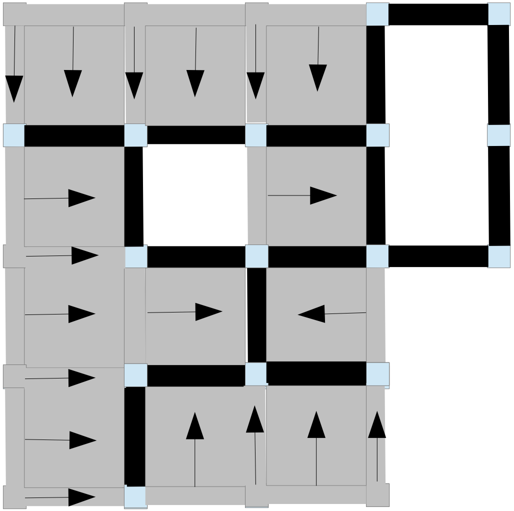
```
Input: vector of cell complex;
  queue T;
  for every a in complex do
    if a has unique coboundary element then
      T.enqueue( a );
  while T is not empty do
    cell a = T.dequeue();
    if a has unique element b in coboundary then

      remove a and b from complex;
      for every c face of b such that c!=a do
        if c has unique coboundary element then
          T.enqueue(c);
      for every c face of a do
        if c has unique coboundary element then
          T.enqueue(c);
```

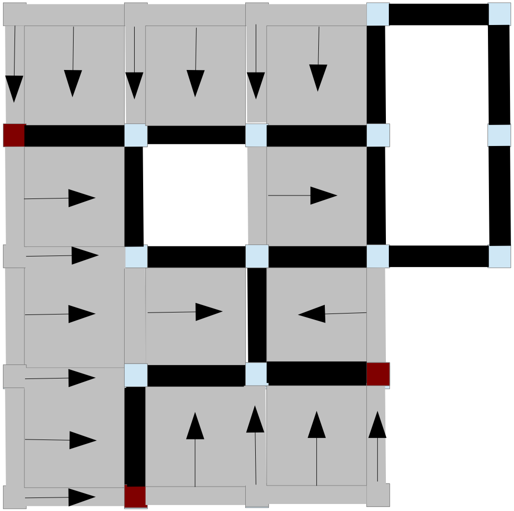
Elementary Reductions.



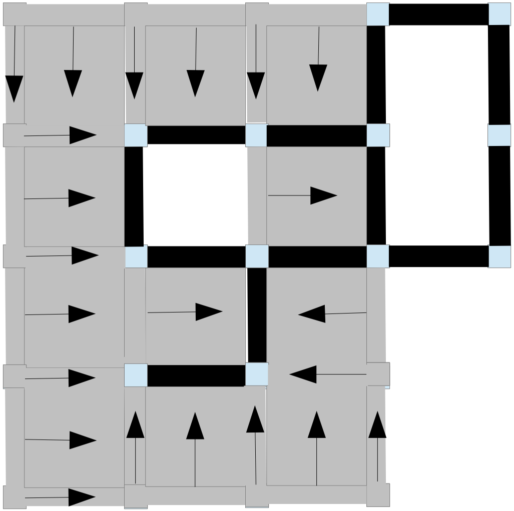
Elementary Reductions.



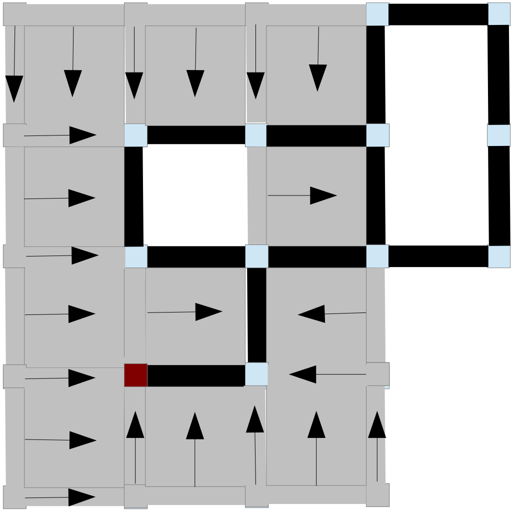
Elementary Reductions.



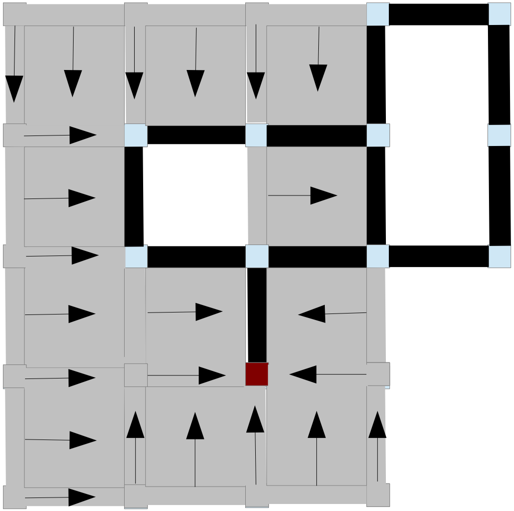
Elementary Reductions.



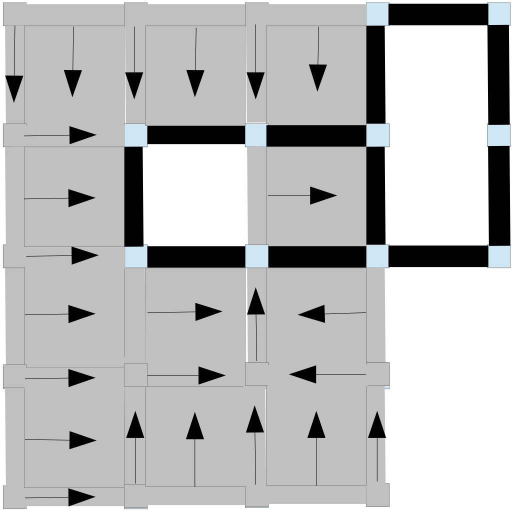
Elementary Reductions.



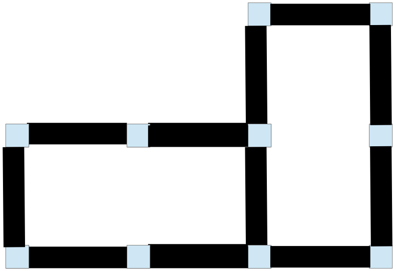
Elementary Reductions.



Elementary Reductions.



Elementary Reductions.



Elementary Reductions.

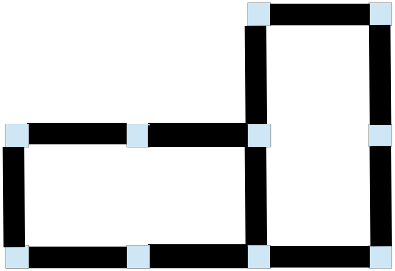
1. Introduced by J. H. C. Whitehead.
2. Homotopy equivalence.
3. Morse paths do not start at any critical cell (they go from boundary to interior).
4. Morse complex is identical to the initial complex minus reduced cells.

Coreductions.

Procedure 5 Coreduction.

```
Input: vector of cell complex;
  queue T;
  Remove a single vertex v from complex;
  for Every c in coboundary of v do
    T.enqueue(c);
  while  T is not empty  do
    c = T.dequeue();
    if c has unique element b in boundary then
      remove c and b from the complex;
      for every element e in coboundary of b do
        if  e has unique element in boundary  then
          T.enqueue(e);
      for every element e in cobounday of c do
        if  e has unique element in boundary  then
          T.enqueue(e);
```

Coreductions.



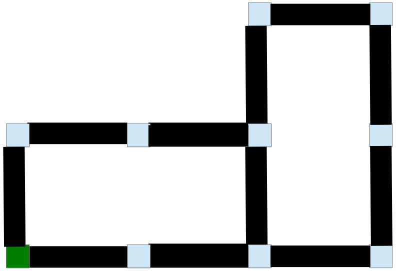
Coreductions.

Procedure 6 Coreduction.

Input: vector of cell complex;

```
queue T;
Remove a single vertex v from complex;
for Every c in coboundary of v do
    T.enqueue(c);
while T is not empty do
    c = T.dequeue();
    if c has unique element b in boundary then
        remove c and b from the complex;
        for every element e in coboundary of b do
            if e has unique element in boundary then
                T.enqueue(e);
        for every element e in cobounday of c do
            if e has unique element in boundary then
                T.enqueue(e);
```

Coreductions.



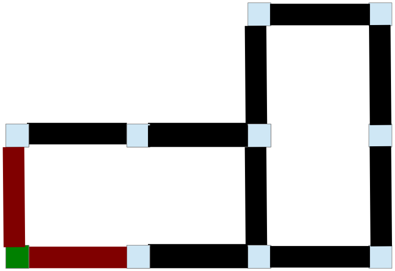
Coreductions.

Procedure 7 Coreduction.

```
Input: vector of cell complex;
  queue T;

  Remove a single vertex v from complex;
  for Every c in coboundary of v do
    T.enqueue(c);
  while T is not empty do
    c = T.dequeue();
    if c has unique element b in boundary then
      remove c and b from the complex;
      for every element e in coboundary of b do
        if e has unique element in boundary then
          T.enqueue(e);
      for every element e in cobounday of c do
        if e has unique element in boundary then
          T.enqueue(e);
```

Coreductions.

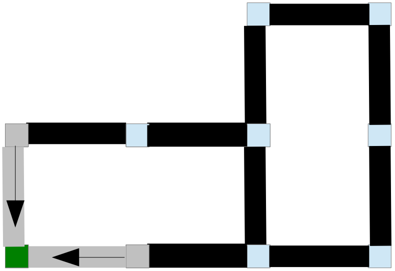


Coreductions.

Procedure 8 Coreduction.

```
Input: vector of cell complex;
  queue T;
  Remove a single vertex v from complex;
  for Every c in coboundary of v do
    T.enqueue(c);
  while T is not empty do
    c = T.dequeue();
    if c has unique element b in boundary then
      remove c and b from the complex;
      for every element e in coboundary of b do
        if e has unique element in boundary then
          T.enqueue(e);
      for every element e in cobounday of c do
        if e has unique element in boundary then
          T.enqueue(e);
```

Coreductions.



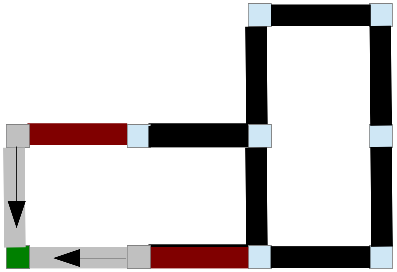
Coreductions.

Procedure 9 Coreduction.

```
Input: vector of cell complex;
  queue T;
  Remove a single vertex v from complex;
  for Every c in coboundary of v do
    T.enqueue(c);
  while T is not empty do
    c = T.dequeue();
    if c has unique element b in boundary then

      remove c and b from the complex;
      for every element e in coboundary of b do
        if e has unique element in boundary
        then
          T.enqueue(e);
      for every element e in cobounday of c do
        if e has unique element in boundary
```

Coreductions.

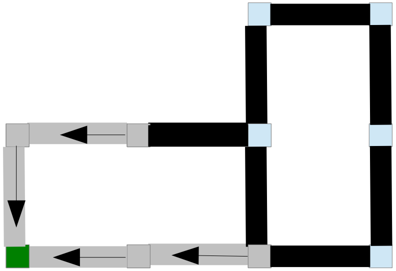


Coreductions.

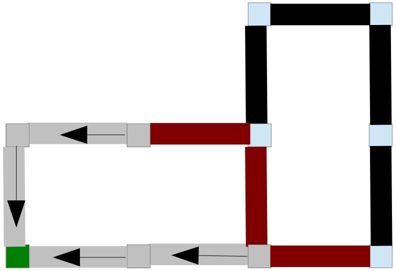
Procedure 10 Coreduction.

```
Input: vector of cell complex;
  queue T;
  Remove a single vertex v from complex;
  for Every c in coboundary of v do
    T.enqueue(c);
  while T is not empty do
    c = T.dequeue();
    if c has unique element b in boundary then
      remove c and b from the complex;
      for every element e in coboundary of b do
        if e has unique element in boundary then
          T.enqueue(e);
      for every element e in cobounday of c do
        if e has unique element in boundary then
          T.enqueue(e);
```

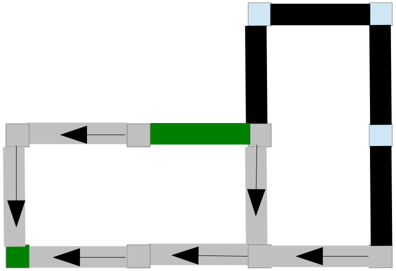
Coreductions.



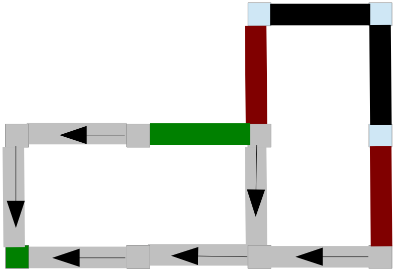
Coreductions.



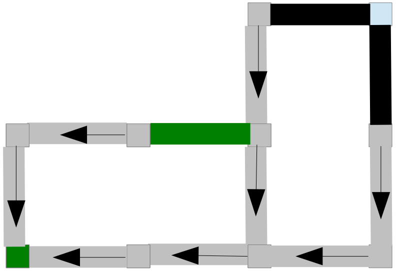
Coreductions.



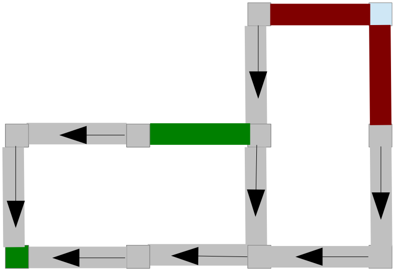
Coreductions.



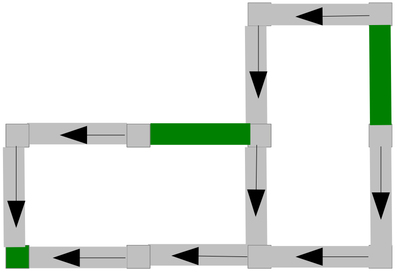
Coreductions.



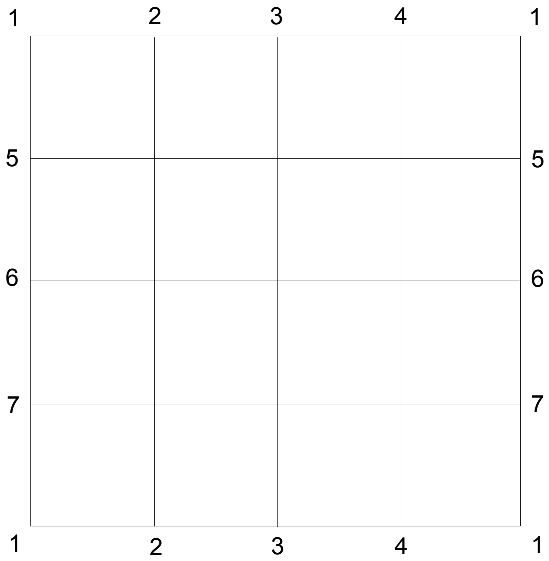
Coreductions.



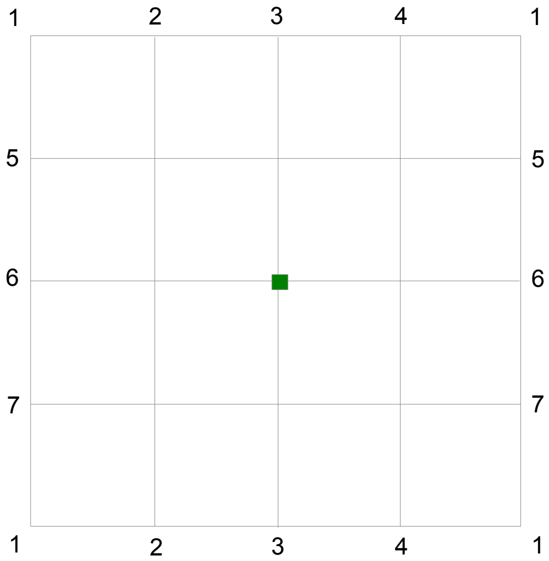
Coreductions.



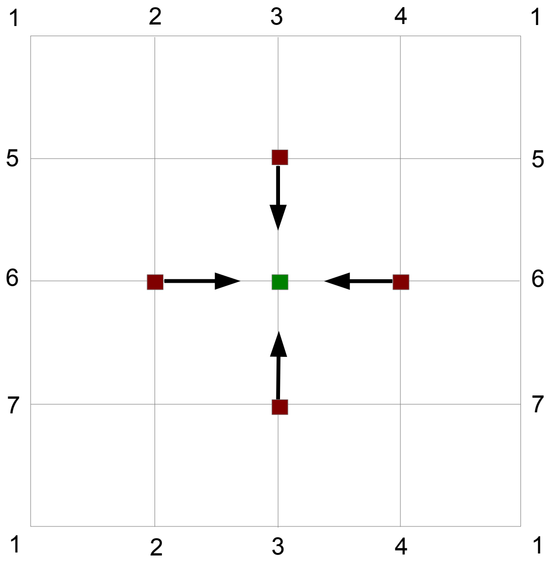
Coreductions of torus.



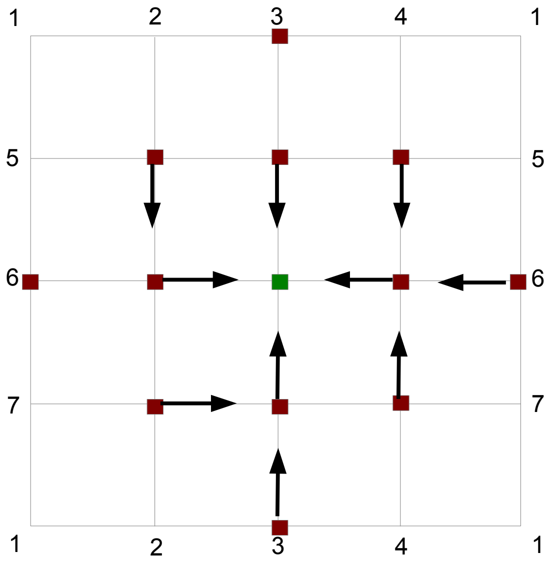
Coreductions of torus.



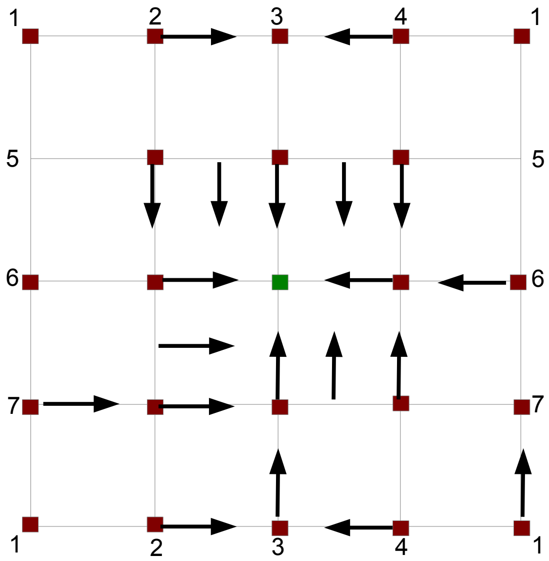
Coreductions of torus.



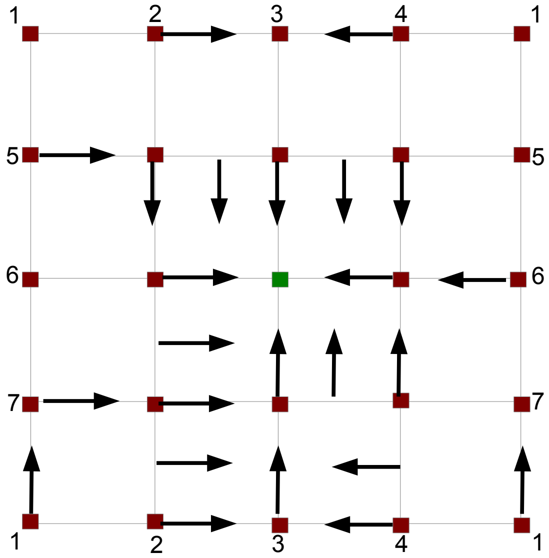
Coreductions of torus.



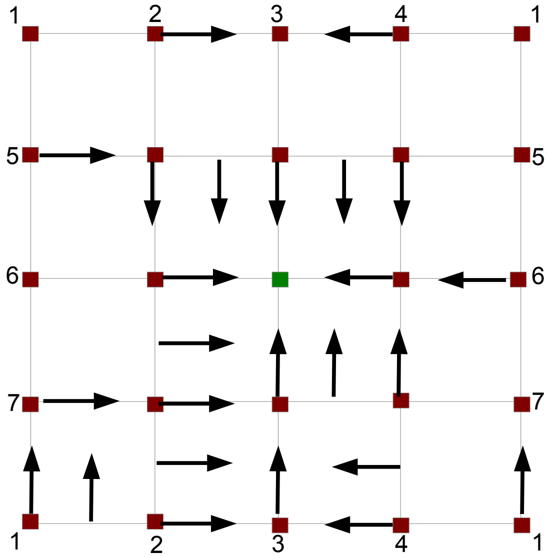
Coreductions of torus.



Coreductions of torus.



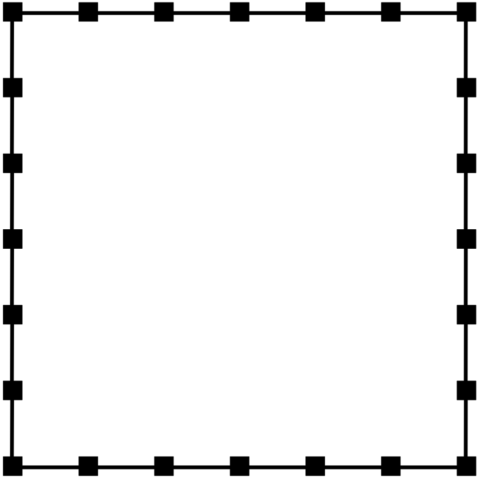
Coreductions of torus.



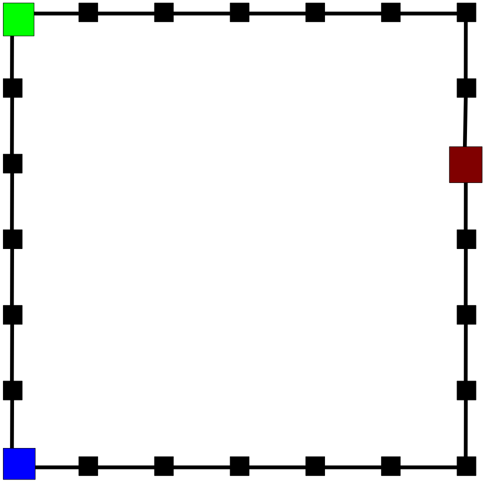
Coreduction.

1. Introduced by Mrozek and Batko.
2. Easy to explain then in language of Discrete Morse Theory.
3. Unique critical (green) point in dimension 0.
4. Gradient field contract everything towards it.
5. Every critical edge has empty boundary.
6. Every gradient path starting at boundary of higher dimension critical cell do not end in any lower dimensional critical cell.
7. Multi-point coreduction (used in parallel programs).

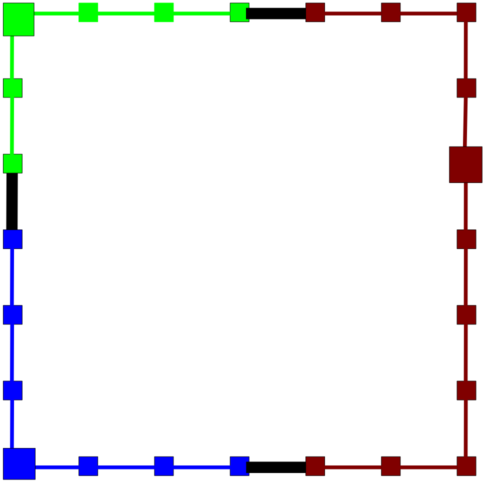
Multi-point coreduction.



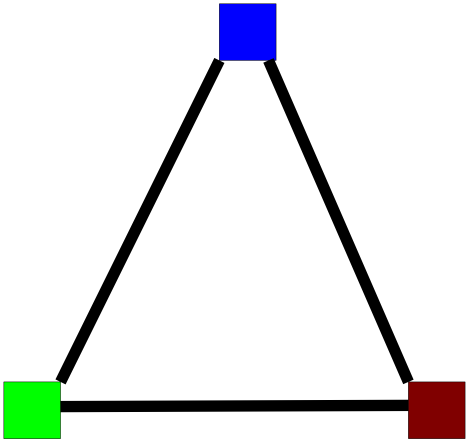
Multi-point coreduction.



Multi-point coreduction.



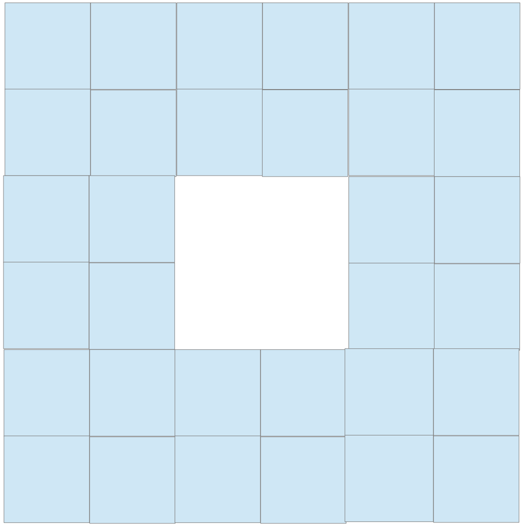
Multi-point coreduction.



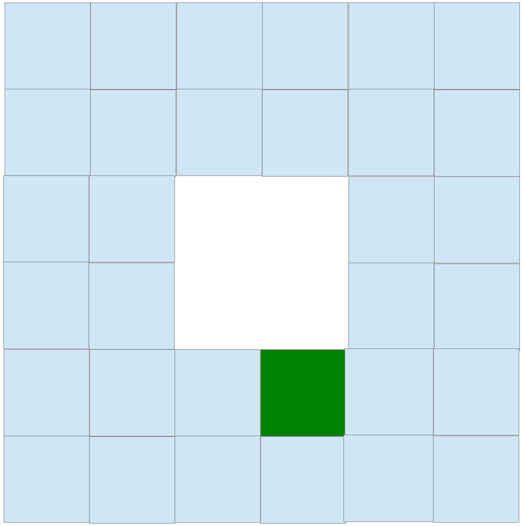
Acyclic subspace.

1. Introduced by Mrozek, Pilarczyk and Zelazna.
2. $A \subset K$ such that $\hat{H}_n(A)$ are trivial (acyclic subcomplex).
3. $H_n(K) \simeq H_n(K, A)$ for $n > 0$ and $H_0(K) \simeq H_0(K, A) \oplus \mathbb{Z}$ (Mayer-Vietoris).
4. $\hat{H}_n(K, A) \simeq \hat{H}_n(K \setminus A)$ (Mrozek, Batko).
5. Greedy strategy to build as large A as possible.

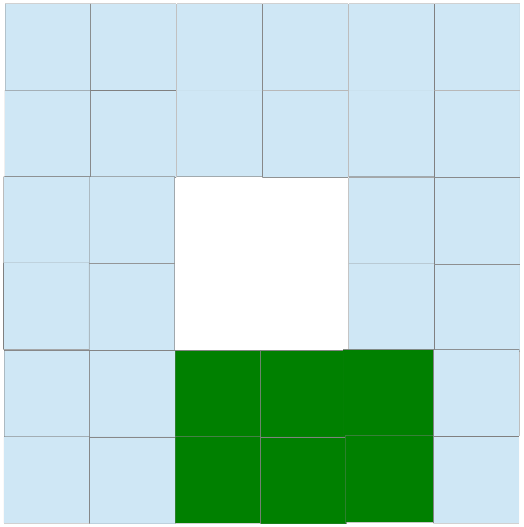
Acyclic subcomplex.



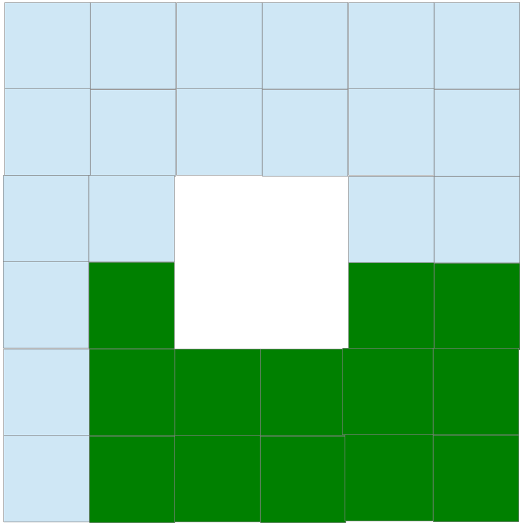
Acyclic subcomplex.



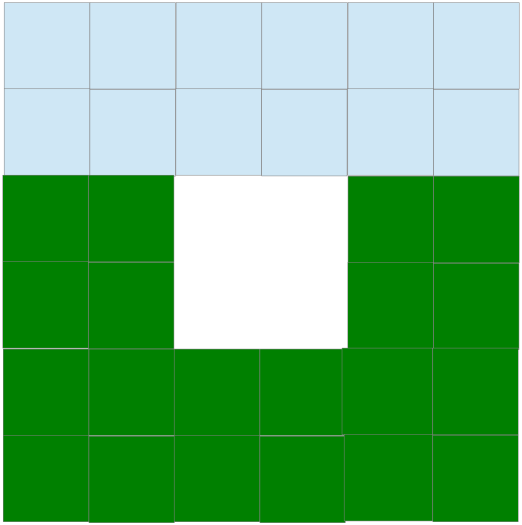
Acyclic subcomplex.



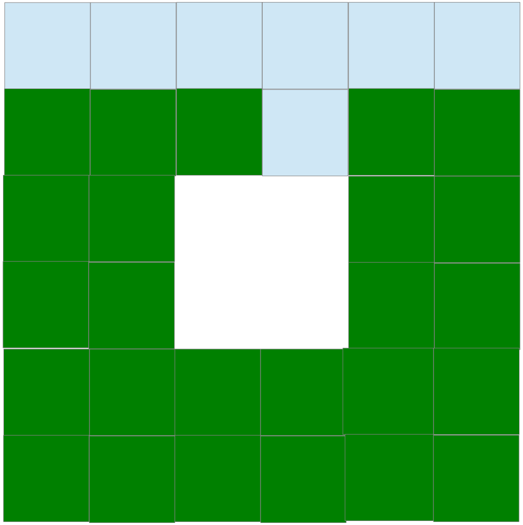
Acyclic subcomplex.



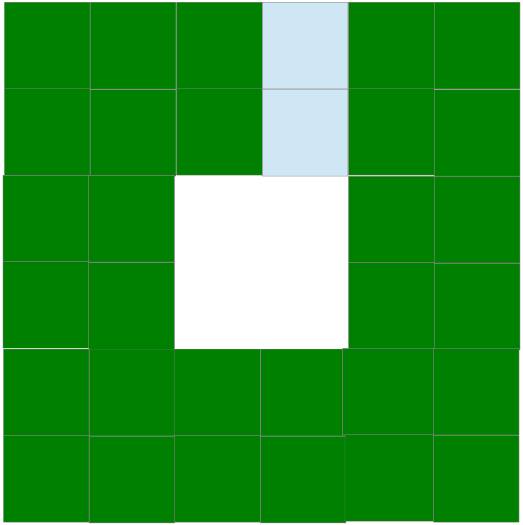
Acyclic subcomplex.



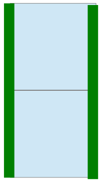
Acyclic subcomplex.



Acyclic subcomplex.



Acyclic subcomplex.



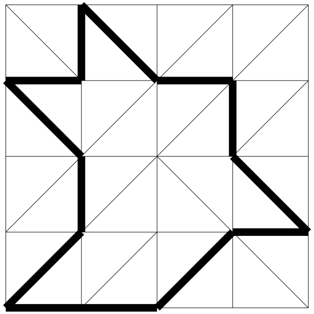
Acyclic subspace.

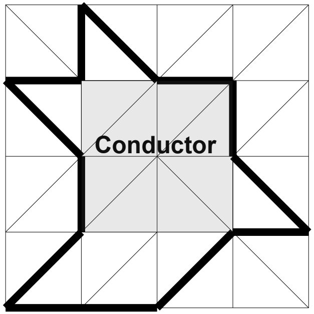
1. This algorithm operates only on top dimensional cells.
2. Can be applied to the basic version of a bitmap.
3. Test needed to determine if an element can be added to acyclic subcomplex.
4. Full test - homology computations, tabulated configurations (2/3d cubes, 2/3/4d simplices).
5. Partial tests – higher dimensions.
6. Coreduction algorithm is also kind of acyclic subspace algorithm.

Retrieving of generators.

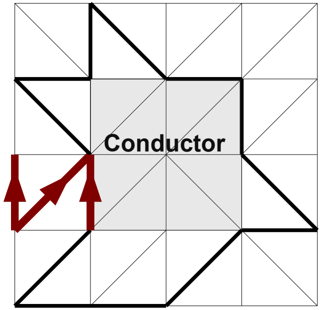
1. In some applications Betti numbers and torsions coefficient are not sufficient.
2. Sometimes representatives of generators are needed.
3. Example of such a application – electromagnetic modeling (we will consider case of Ampere's law).
4. Local Ampere's law: For every face f, $\langle EM, \partial f \rangle = \langle Current, f \rangle$.

Ampere's law.

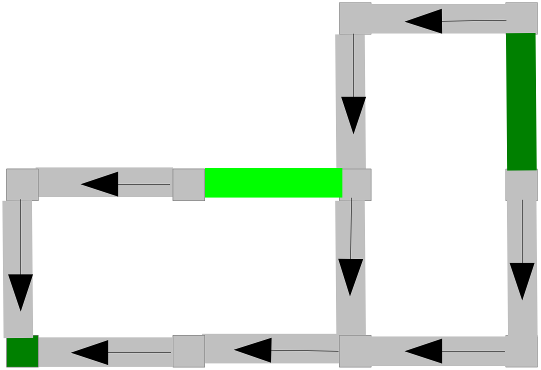




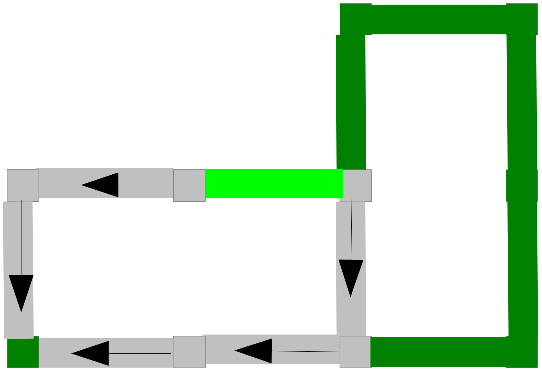
Ampere's law.



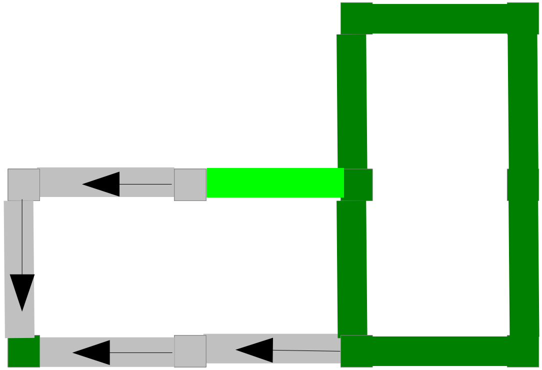
Retrieving generator.



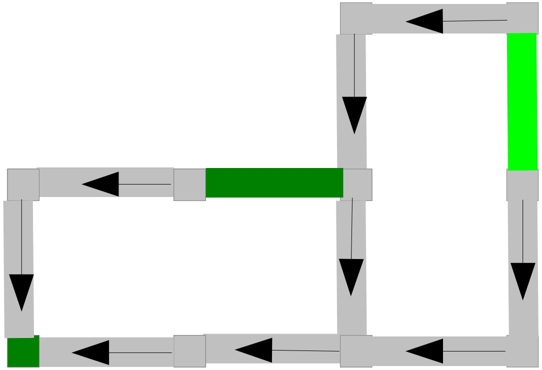
Retrieving generator.



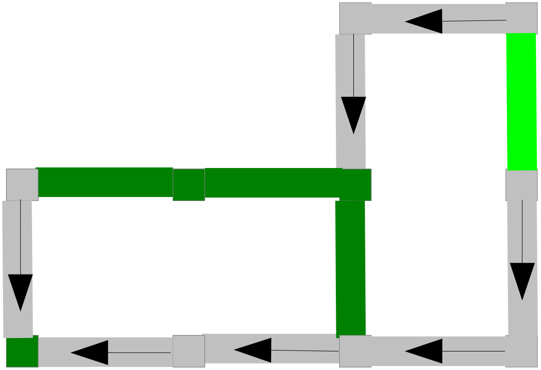
Retrieving generator.



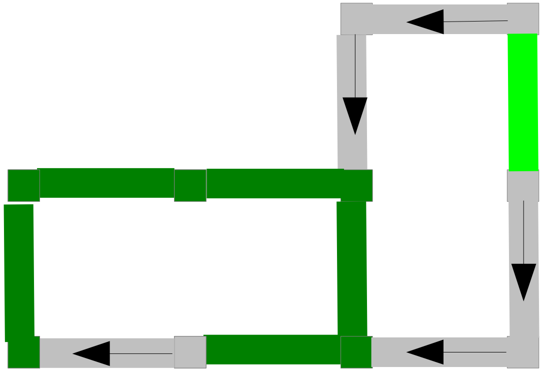
Retrieving generator.



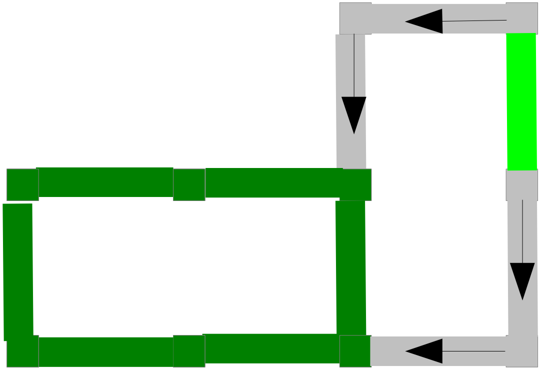
Retrieving generator.



Retrieving generator.



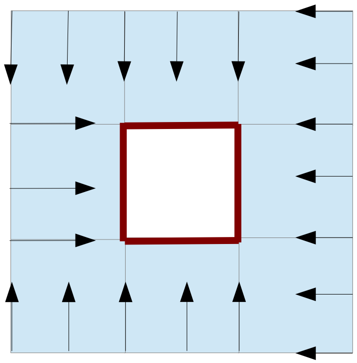
Retrieving generator.



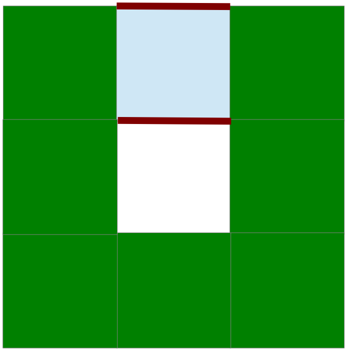
Retrieving of generators.

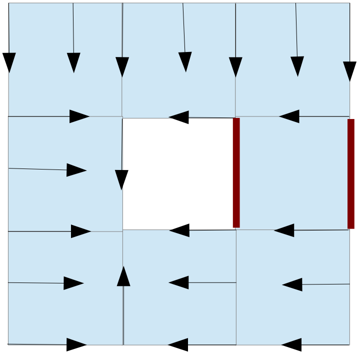
1. Restoring generator takes $O(n)$ time per generator.
2. The constant is considerable.
3. It pays off to use so called *shaving* as a first reduction.
4. *Shaving* is a reduction such that there is no need to restore generators...
5. i.e. embeddings of generators in reduced complex are generators in initial complex.
6. Elementary reductions is a shaving for homology.
7. Acyclic subspace and coreductions are shaving for cohomology.
8. For EM computations shaving gives about order of magnitude speed up in computations.

Shaving for homology.



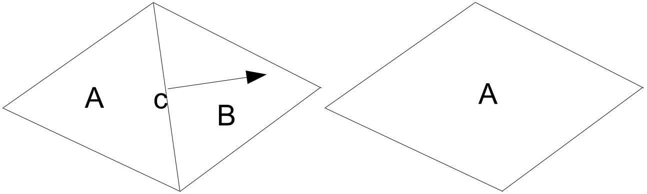
Shaving for cohomology.





KMS - algebraic reductions.

1. Algebraic reduction – bases on making a single Morse pairing at a time:



2. $\partial A = \partial A - \langle \partial B, c \rangle^{-1} \langle \partial A, c \rangle \partial B$ provided $\langle \partial B, c \rangle$ is invertible.
3. Such a reduction is applied iteratively.

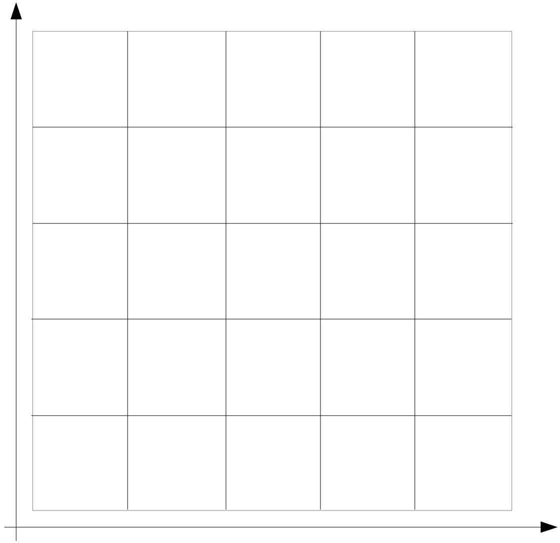
What to do when the reductions are done?

1. Reductions are heuristics.
2. They works very well in the applications we are considering (dynamical systems, engineering, material analysis).
3. They give a Betti numbers and generators in planar and 2-manifold case (no SNF needed).
4. For some algebraic complexes they works much worst.
5. To obtain integer homology at the end SNF algorithm is needed.
6. For field homology and persistence one can use also iterated Morse approach.

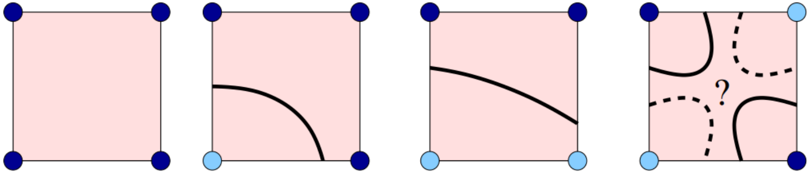
Homology of nodal domains of continuous functions.

1. There are rigorous partial algorithms to compute homology of nodal domains of a function.
2. A function $u : \mathbb{R}^2 \to \mathbb{R}$ is given as a computer program (formula, numerical method) ...
3. such that it can be evaluated with interval arithmetic.
4. With this information topologically faithful representation of nodal domains can be obtained.

Initial discretization of the domain.



Initial discretization of the domain.

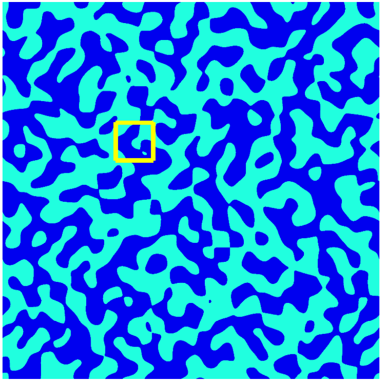


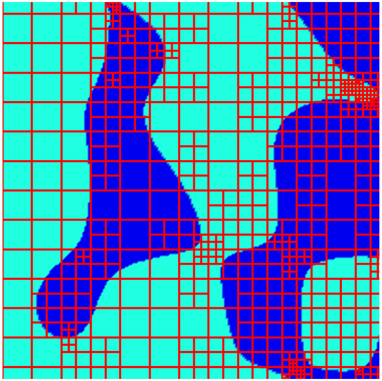
Validation algorithm.

1. For every rectangle R...
2. make sure that in each corner $x \in R$ the value of $u(x) \neq 0$. If it is, then *failure*.
3. If all the signs are the same, then compute $[\underline{u}(R), \bar{u}(R)]$. If it contain zero, then subdivide. Otherwise OK.
4. If all but one signs are the same, then if either $u_x(R)$ or $u_y(R)$ contain zero, then subdivide. Else OK.
5. If two corners (upper/lower) are positive and other two (lower/upper) are negative then if:
   5.1 u on lower and upper edge does not contain zero and
   5.2 $u_x(R)$ does contain zero

   then OK, else subdivide. The same in case of left/right.

Nodal domains of random trigonometric polynomial.

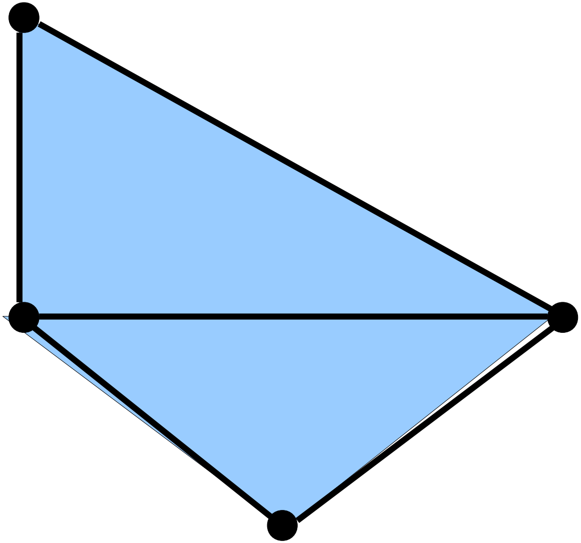




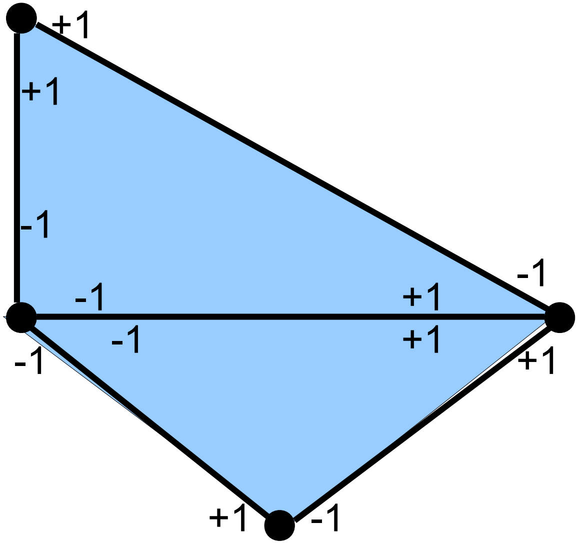
How to compute homology of non-regular cubical grid?

1. Subdivision to cubical complex.
2. Homology of Čech complex.
3. Computational homology theory for regular CW-complexes.
4. The key point is to obtain incidence coefficients – Massey's equations.

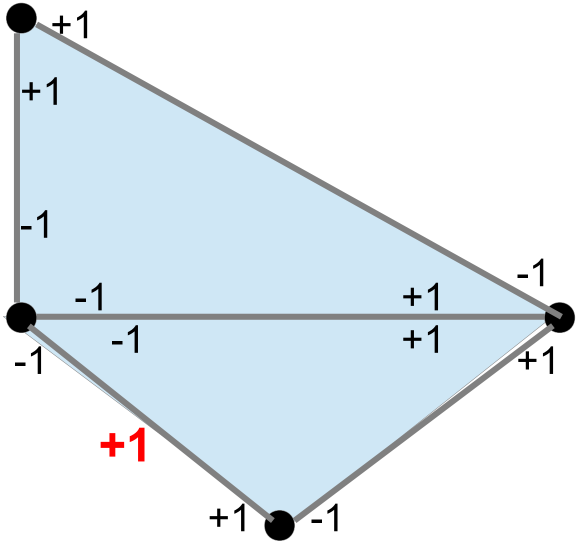
Massey's equations.



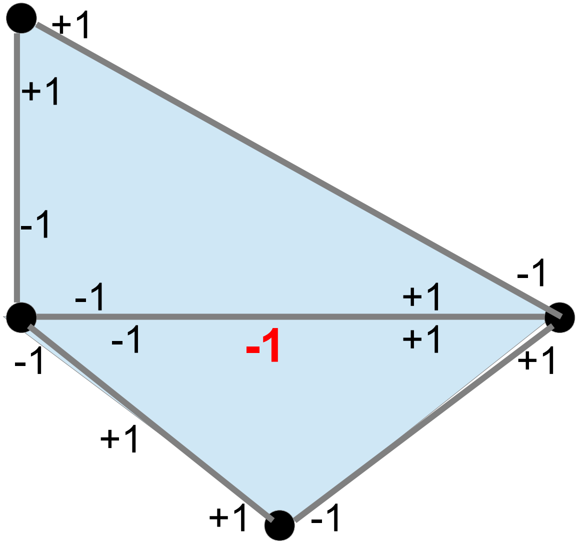
Massey's equations.



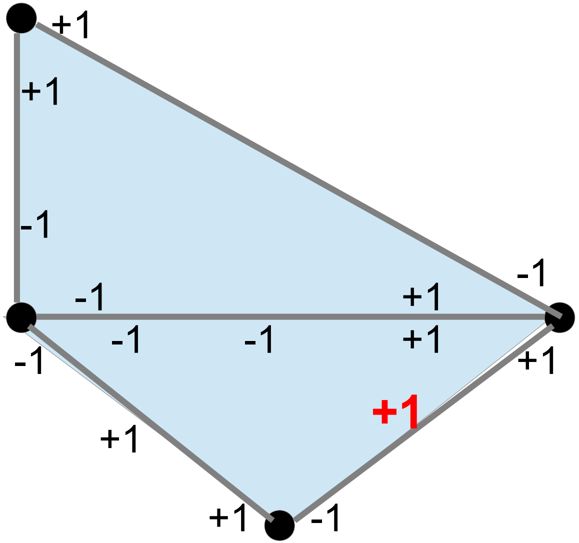
Massey's equations.



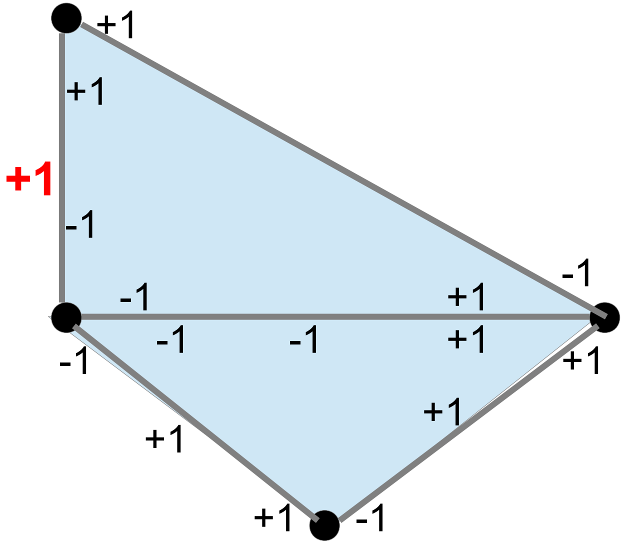
Massey's equations.



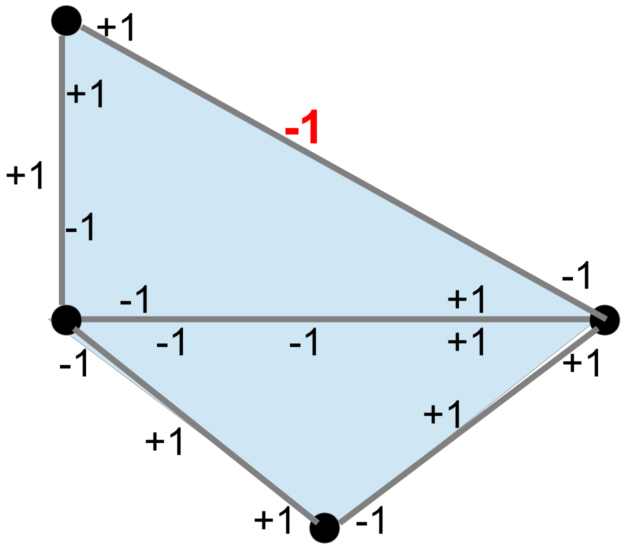
Massey's equations.



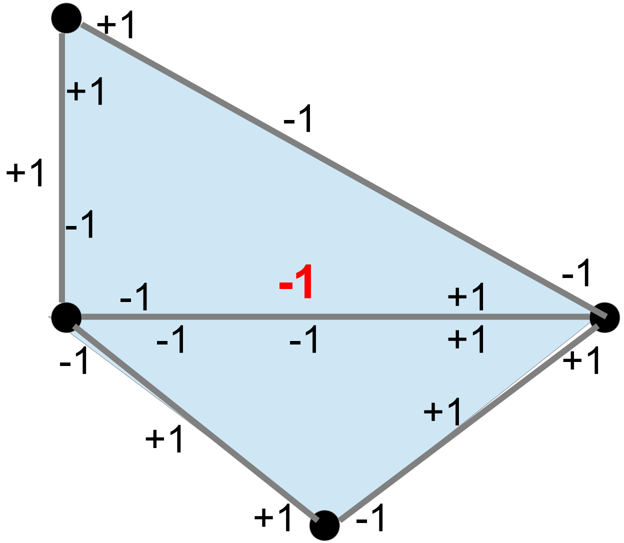
Massey's equations.



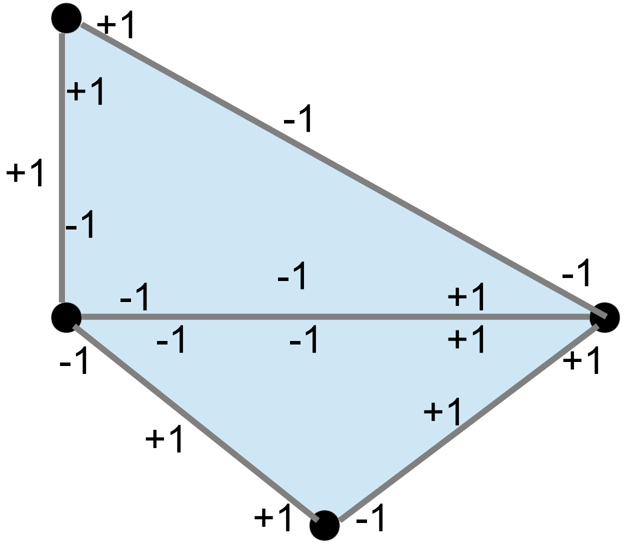
Massey's equations.



Massey's equations.



Massey's equations.



Once we have incidence coefficients.

1. Any presented reduction method can be used.
2. In case of random trigonometric polynomials the memory used by the pointer structure is order(s) of magnitude lower than by corresponding uniform cubical grid.
3. Also, computational times are $\sim$ two orders of magnitude better.
4. But this is what we see in practice for random trig. polynomials.
5. Intuition - if the 'shape' of nodal domain is not very complicated, then the pointer representation is superior.
6. If it become ver complicated / chaotic / fractal, then regular cubical grid may be better.

Delfinado-Edelsbrunner incremental algorithm.

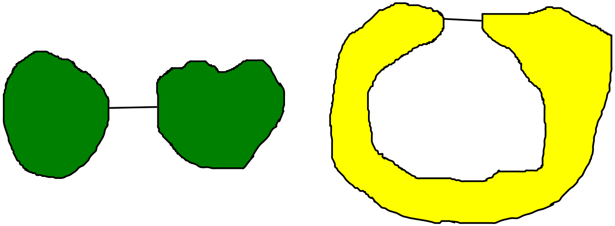
1. It is a first algorithm to compute homology different from standard SNF algorithm.
2. It works for sub-triangulations of S^3.
3. Bases on Alexander duality – X - sub-triangulation of S^3, $Y = S^3 \setminus X$.
4. $H_q(Y) \simeq H^{2-q}(X)$. Since there are no torsions in $\mathbb{R}^3 \implies \beta_q(Y) = \beta_{2-q}(X)$.
5. $2-$cycles in X are 1-1 to connected components in Y.

Delfinado-Edelsbrunner incremental algorithm.

1. $\sigma_1, \sigma_2, \ldots, \sigma_n$ triangulation of S^3 such that every prefix is a simplicial complex.
2. $X_i = \{\sigma_1, \sigma_2, \ldots, \sigma_i\}$.
3. Suppose σ_{i+1} of dimension d is added to X_i. Then either $\beta_d(X_{i+1}) = \beta_d(X_i) + 1$ or $\beta_{d-1}(X_{i+1}) = \beta_{d-1}(X_i) - 1$.
4. If σ_{i+1} is a vertex, then always $\beta_0(X_{i+1}) = \beta_0(X_i) + 1$.
5. If σ_{i+1} is edge or triangle, then we should determine if it closes a cycle.
6. If it does, $\beta_{dim(\sigma_{i+1})}$ increase, if not $\beta_{dim(\sigma_{i+1})-1}$ decrease.

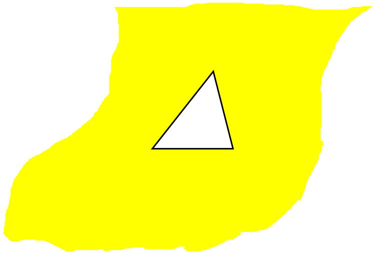
Detecting 1-cycles.

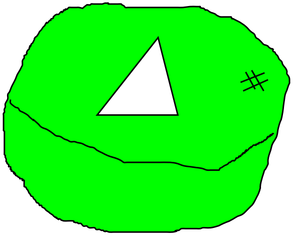
1. Analyzing connected components of 1-skeleton.



Detecting 2-cycles.

1. Analyzing connected components of complex complement.





Delfinado-Edelsbrunner incremental algorithm.

Procedure 11 Incremental Algorithm (non-optimal version!).

```
Input: $\sigma_1, \dots, \sigma_n$ filtration of $S^3$;
  $\beta_0 = \beta_1 = \beta_2 = \beta_3 = 0$;
  for $i = 1$ to n do
    if $\sigma_i$ is a vertex then
      $\beta_0 + +$;
    if $\sigma_i$ is an edge then
      if $\sigma_i$ close a cycle in $X_{i-1}$ then
        $\beta_1 + +$;
      else
        $\beta_0 - -$;
      if $\sigma_i$ is a triangle then
        if $\sigma_i$ close a cycle in $S^3 \setminus X_i$ then
          $\beta_2 - -$;
        else
          $\beta_2 + +$;
    if $i = n$ then
      $\beta_3 = 1$;
```

Bibliography.

1. Data structures:
   1.1 T. Kaczynski, K. Mischaikow, and M. Mrozek, "Computational Homology" (book).
   1.2 H. Edelsbrunner and J. Harer, "Computational Topology" (book).
   1.3 P.D., T. Kaczynski, M. Mrozek, Th. Wanner, "Coreduction Homology Algorithm for Regular CW-Complexes".
   1.4 Chomp – Computational Homology Project (soft).
   1.5 CAPD – Coputer Assisted Proofs in Dynamic (soft).
   1.6 Soft for persistence (ancient mythology) – Dionysus (D. Morozov), Perseus (V. Nanda),
   1.7 Soft for persistence (rest) – (J)Plex (Stanford), Homology with a Twist (IST Austria).

Bibliography.

1. Reduction methods:
   1.1 T. Kaczynski, K. Mischaikow, and M. Mrozek, "Computational Homology" (book).
   1.2 R. Forman, "A user's guide to discrete Morse theory".
   1.3 T. Kaczynski M. Mrozek and M. Slusarek, "Homology Computation by Reduction of Chain Complexes".
   1.4 M. Mrozek, P. Pilarczyk, N. Zelazna, "Homology Algorithm Based on Acyclic Subspace".
   1.5 M. Mrozek, B. Batko, Coreduction Homology Algorithm.
   1.6 P. Brendel, P. D., M. Mrozek, N. Zelazna, "Homology Computations via Acyclic Subspace".

Bibliography.

1. Cohomology computations:
    1.1 P. D., R. Specogna "Efficient cohomology computation for electromagnetic modeling".
2. Homology of nodal domain:
    2.1 S. Day, W. Kalies, Th. Wanner ,"Verified homology computations for nodal domains".
    2.2 P.D., T. Kaczynski, M. Mrozek, Th. Wanner, "Coreduction Homology Algorithm for Regular CW-Complexes".
    2.3 G. Cochran, Th. Wanner, P.D., "A randomized subdivision algorithm for determining the topology of nodal sets".
    2.4 W. S. Massey, "A Basic Course in Algebraic Topology" (book).
3. Delfinado-Edelsbrunner (Incremental Algorithm):
    3.1 C. Delfinado, H. Edelsbrunner, "An incremental algorithm for Betti numbers of simplicial complexes".
    3.2 H. Edelsbrunner and J. Harer, "Computational Topology" (book).

Thank you for your time!

Pawel Dlotko
University of Pennsylvania
dlotko@sas.upenn.edu
pawel_dlotko @ skype
pdlotko @ gmail