COMPUTATIONAL METHODS IN SCIENCE AND TECHNOLOGY **12(2)**, 115-121 (2006)

Homology Calculation of Cubical Complexes in $\mathbb{R}^n$

Piotr Kot

Institute of Mathematics, Cracow University of Technology
Warszawska 24, 31-155 Kraków, Poland
e-mail: pkot@usk.pk.edu.pl

(Rec.11 November 2005)

Abstract: The purpose of this article is to present an algorithm based on the simplification of the boundaries and coboundaries of cubes in $\mathbb{R}^n$. We extend the algorithm of W. D Kalies, K. Mischaikow and G. Watson which can be found in the paper *Cubical approximation and computation of homology* published in Banach Center Publ. **47**, 115-131 (1999).

Key words: homology calculation

1. INTRODUCTION

The standard algorithms for computing the homology groups of cellular complexes composed of cubes (cubical sets) are based on performing row and column operations on the boundary matrices. We present the algorithm which is an extension of the results presented in [1]. The main idea is based on [1], but the construction of particular procedures and their theoretical analysis is new. Owing to the specific analysis of the problem which is presented in this paper, final estimations are even better than those in [1].

The algorithm obtained allows to compute the homology groups for two-dimensional cubical sets. In higher dimensions, it sometimes does not lead to the computing the specified groups. However, it can lead to essential simplification of basic complex (cubical set). Moreover, we conclude that the homology groups for two-dimensional cubical sets can be calculated in $O(N \log^2(N))$ operations, where N is the number of cubes in the considered complex. Whereas in [1] the corresponding time is $O(N \log^3(N))$.

Let n natural number greater than 1. We consider logarithms of the base 2. Let ∂B the list of all cells which belong to the boundary of B and δB the list of all cells which belong to the coboundary of B.

2. COMPUTATIONAL HOMOLOGY

In this section we describe a theoretical background for computational homology. We use terminology from the book [2].

2.1. Cubical sets

For a given natural number $a \in \mathbb{N}$ we say that $[a, a+1]$ and $[a, a]$ are elementary intervals. Now we say that a set $X \subset \mathbb{R}^n$ is cubical if X can be written as a finite union of elementary cubes: $X = \bigcup_{i=1}^{m} Q_i$ where Q_i is a finite product of elementary intervals I_i^k:

$$Q_i = I_i^1 \times \cdots \times I_i^k \times \cdots \times I_i^n .$$

The dimension of an elementary cube Q is defined to be the number of nondegenerate intervals in definition of Q. We say that a cube Q is k-cube if dimension of Q is equal k. Let $\mathcal{K}_k^d$ be the set of all k-cubes contained in $\mathbb{R}^d$.

2.2. Cubical chains

With each elementary k-cube $Q \in \mathcal{K}_k^d$ we associate an algebraic object $\hat{Q}$ called an elementary k-chain of $\mathbb{R}^d$. The set of all elementary k-chains of $\mathbb{R}^d$ is denoted by $\hat{\mathcal{K}}_k^d$. Now we define k-chain c as a finite sum of elementary k-chains. In particular $c = \sum_{i=1}^{s} \alpha_i \hat{P}_i$ where $\alpha_i = \mathbb{Z}$ and $\hat{P}_i \in \hat{\mathcal{K}}_k^d$. The set of all k-chains of $\mathbb{R}^d$ we denote as C_k^d. If $X \subset \mathbb{R}^d$ is a cubical set then by $C_k(X)$ we denote the set of all k-chains generated by elementary k-cubes contained in X.

2.3. The boundary operator

We would like to define the cubical boundary operator which takes k-chains to $(k-1)$-chains. This is the motivation for defining the following product:

For given two elementary cubes $P \in \mathcal{K}_{k_1}^{d_1}$ and $Q \in \mathcal{K}_{k_2}^{d_2}$ we set

$$\hat{P} \lozenge \hat{Q} = \widehat{P \times Q}.$$

This definition extends to arbitrary chains

$$c_1 = \sum_{i=1}^{s_1} \alpha_i \hat{P}_i \in C_{k_1}^{d_1}$$

and

$$c_2 = \sum_{j=1}^{s_1} \beta_j \hat{Q}_j \in C_{k_2}^{d_2}$$

by

$$c_1 \diamond c_2 := \sum_{ij} \alpha_i \beta_j \widehat{P_i \times Q_j}.$$

Now for a given $k \in \mathbb{N}$ we can define the cubical boundary operator

$$\partial_k : C_k^d \to C_{k-1}^d,$$

First we define ∂_k on an elementary chain $\hat{Q} \in \hat{\mathcal{K}}_k^d$ by the induction on the given number d as follows.

If $d = 1$ then Q is an elementary interval and hence $Q = [a, a] \in \mathcal{K}_0^1$ or $Q = [a, a+1] \in \mathcal{K}_1^1$ for some $a \in \mathbb{N}$.

Define

$$\partial_k \hat{Q} := \begin{cases} 0 & \text{if} \quad Q = [a, a] \\ \widehat{[a, a+1]} - \widehat{[a, a]} & \text{if} \quad Q = [a, a+1] \end{cases}.$$

Now assume that $d > 1$. Then $Q = I \times P$ where I is an elementary interval and P an elementary cube.

Define

$$\partial_k \hat{Q} = \partial_{k_1} \hat{I} \diamond \hat{P} + (-1)^{k_1} \hat{I} \diamond \partial_{k_2} \hat{P},$$

where k_1 is a dimension of I and k_2 is a dimension of P.

Finally, we extend the definition to all chains by linearity; that is, if $c = \sum_{i=1}^{m} \alpha_i \widehat{Q_i}$ then

$$\partial_k c := \sum_{i=1}^{m} \alpha_i \partial_k \widehat{Q_i}.$$

2.4. Homology of cubical sets

Let $X \subset \mathbb{R}^d$ be a cubical set. A k-chain $z \in C_k(X)$ is called a k-cycle if $\partial_k z = 0$. The set of all k-cycles in X is denoted as

$$Z_k(X) := \ker \partial_k^X = C_k(X) \cap \ker \partial_k \subset C_k(X).$$

A k-chain $z \in C_k(X)$ is called a k-boundary in X if there exists $c \in C_{k+1}(X)$ such that $\partial_{k+1} c = z$. The set of all k-boundaries in X is denoted as

$$B_k(X) := \operatorname{im} \partial_{k+1}^X = \partial_{k+1}\left(C_{k+1}(X)\right) \subset C_k(X).$$

It can be proved that $B_k(X) \subset Z_k(X)$.

Now we can define k-th cubical homology group of X as the quotient group:

$$H_k(X) := Z_k(X) / B_k(X).$$

The homology of X is the collection of all homology groups of X:

$$H_*(X) := \{H_k(X)\}_{k \in N}.$$

Note that we use the homology groups of the cubical set X to gain information about the topological structure of X.

1.5. Fixed points in the unit ball

In this section we give the application of computational homology.

Let $\mathbb{B}^{n+1} = [-1, 1]^{n+1}$ be the unit ball and $\mathbb{S}^n = \partial \mathbb{B}^{n+1}$ be the corresponding unitary sphere.

First we need the following fact about Homotopy Invariance:

Corollary 1. [2, Corollary 6.69]. *Let X and Y be cubical sets of the homotopy type. Then*

$$H_*(X) \cong \{H_*(Y)\}.$$

We have the following theorem due to K. Borsuk:

Theorem 1. [2, Theorem 10.43] *Let* $n \geq 0$. *The following statements are equivalent:*

(1) $\mathbb{S}^n$ *is not contractible.*

(2) *Every continuous map* $f : \mathbb{B}^{n+1} \to \mathbb{B}^{n+1}$ *has a fixed* point.

(3) *There is no retraction from* $\mathbb{B}^{n+1}$ *onto* $\mathbb{S}^n$.

Now we can use a computer program to calculate homology groups:

For $d \geq 1$ we can obtain:

$$H_k(\mathbb{S}^n) \cong \begin{cases} \mathbb{Z} & \text{if } k = 0, d \\ 0 & \text{otherwise} \end{cases}$$

while for $d = 0$

$$H_k(\mathbb{S}^n) \cong \begin{cases} \mathbb{Z}^2 & \text{if } k = 0 \\ 0 & \text{otherwise} \end{cases}$$

Observe that homology groups for a point $\{0\}$ are the following:

$$H_k(\{0\}) \cong \begin{cases} \mathbb{Z} & \text{if } k = 0, d \\ 0 & \text{otherwise} \end{cases}.$$

In particular due to Corollary 1 we may conclude that $\mathbb{S}^n$ is not contractible to a point $\{0\}$, thus the condition (a) of Theorem 1 holds, and therefore all three statements of Theorem 1 are true.

More examples can be found in the book (2).

3. CUBICAL APPROXIMATIONS

```
Algorithm 1: Subdivide(B,K,s)
input : solid B, cube K, paraKaliesmeter s
output: The tree which approximates solid B by means of leaves chain

begin
    if $s > 0$ then
        Divide K into $2^n$ of identical cubes: $K_1, ..., K_{2^n}$ sorted lexicographically due
        to coordinates;
        for $i = 1$ to $2^n$ do Subdivide($B, K_i, s-1$);
    end
    else
        if $B$ crosses $K$ then send 1 to output;
        if $B$ does not cross $K$ then send 0 to output;
    end
end
```

We observe that the cube $[a, b]^n$ where $a < b$ can be divided into 2^n cubes $[a_1, b_1] \times ... \times [a_n, b_n]$ where $a_i = a$ and $b_i = (a + b)/2$ or $a_i = (a + b)/2$ and $b_i = b$. From here comes a clear method of approximation any solid B contained in cube K with by means of calling the procedure Subdivide (K, B, s).

4. ALGORITHM FOR COMPUTING THE HOMOLOGY

In this section we describe the algorithm for computing the homology groups for the complexes composed of cubes in $\mathbb{R}^n$ (cubical sets). Simplification method of two given cells is the same as in the paper [1]. A crucial difference between the algorithm in our paper and the one in [1] is different order in which the cells are selected for simplification (this order is set by a special index which stores information on the simplifications already carried out near the selected cells).

4.1. Data structure

The algorithm operates upon the following data structures:

- Leaf $L \Leftrightarrow L$ is the leaf in the tree returned by Subdivide(K, B, s).
- LeafParent $T \Leftrightarrow$ if Q is a child for T then Q is a leaf in the above tree. Moreover all children of T are d as Children(T).
- Cell $X \Leftrightarrow X$ contains coordinates, boundary list ∂X and a list of coboundaries δX. Additionally, X includes the field Index: Index(X).
- BoCell $Y \Leftrightarrow Y \in \partial X$ where X is of type Cell. In particular Y is a simple pointer for a suitable cell of type Cell of lower dimension.
- CoCell $Z \Leftrightarrow Z \in \delta X$ where X is of type Cell. Now we define an incidence number $\langle Z, X \rangle$. In particular Z is the pointer for an appropriate cell of type Cell of higher dimension plus incidence number. If $\langle A, B \rangle \neq 0$ then $B \in \partial A$ and $A \in \delta B$.
- Block $T \Leftrightarrow T$ is a sorted lists[1] (due to Index, dimension and coordinates) of the cells of type Cell. The d-dimensional interior of that cube is d by Interior(T, d). In particular $A \in$ Interior(T, d) if $A \in T$ and A is d-dimensional Cell located outside the boundary of T.

Remark. We denote length of ∂b by $\#\partial b$ and length of δb by $\#\delta b$. Inserting and deleting operations for elements from ∂b and δb are performed in logarithmic time.

4.2. Procedures

Now we present all the procedures.

```
Algorithm 2: CreateCube(L)
input : Leaf L
output: Block B related to L

begin
    Create Block B in the position of Leaf L;
    /*We create Cells, CoCells and BoCells                          */
end
```

```
Algorithm 3: Concatenate($U_1, ..., U_{2^n}$,W)
input : Block $U_1, ..., U_{2^n}$ from level $p$
output: Block W from level $p - 1$

begin
    Block W $= \emptyset$;
    foreach $i := 1, ..., 2^n$ and $A \in U_i$ do
        if $A$ has the same coordinates and dimension as $B$ belonging $W$ then
            $\delta B := \delta B \cup \delta A$;
            /*we treat these two sets as sorted, we unite them and later we create
              an adequate tree                                       */
            Index($B$) := max{Index($B$), Index($A$)};
            U := U \ {A};
        else
            W := W ∪ {A}
        end
    end
end
```

```
Algorithm 4: SetIndex(b,y)
input : Cell b, Cell y
output: Index($y$) is changed due to Index($b$)

begin
    if $b \neq y$ and Index($y$)=Index($b$) then
        Index($y$):=Index($y$)+1;
        else Index($y$) := max{Index($y$), Index($b$)}
    end
end
```

[1] For Concatenate operations we need to merge lists due to coordinates and dimensions. Therefore it suffices to use sorted list. For Simplify operations we additionally need the field Index and therefore we need T to be represented by priority queue.

Algorithm 5: Simplify(W,d)

```
input  : Block W, dimension d
output: Block W is simplified so that Interior(W,d) does not contain cells of
        dimension d with incidence numbers equal to 1,-1

begin
    foreach $b \in Interior(W,d)$ do
        /*We take b with the smallest possible Index */
        if $c \in \delta b$ with $|\langle c, b\rangle| = 1$ then
            Define $X := \delta b \setminus \{c\}$, $Y := \partial c \setminus \{b\}$;
            foreach $x \in X$ do
                replace b in $\partial x$ with Y
                SetIndex($b,z$) for all $z \in \partial x$;
            end
            foreach $y \in Y$ do
                SetIndex($b,y$);
                add X to $\delta y$
                $\langle x, y\rangle := \langle x, y\rangle - \langle c, b\rangle\langle x, b\rangle\langle c, y\rangle$ for all $x \in X$;
            end
            delete b, c ;
        end
    end
end
```

Note 1. Index change in lines 7, 10 influences the order of processing the elements in line 2. However, it does not influence the cost of performing the algorithm as index is not changed into smaller one (priority queue due to Index may be used here). Additionally, for $n = 2$ we can use the fact that Index is of order $O(\log N)$.

Algorithm 6: Main

```
input  : The tree which approximates a given solid by means of leaves chain
output: Homologically equivalent object which consists of the list of cells with
        incidence numbers different from 1,-1.

begin
    foreach LeafParent T in the tree do
        foreach Leaf L ∈ Children(T) do
            CreateCube(L);
        end
        Let $U_1, ..., U_{2^n}$ be Blocks which are children for T ;
        Concatenate($U_1, ..., U_{2^n}$,W);
        /*the above line creates Block W in T position */
        foreach d from 0 to n-1 do Simplify(W,d);
        Delete $U_1, ..., U_{2^n}$ and elements from Children(T);
    end
    Let Root be Block in root position in the tree.;
    Interior(Root,*)=Root ;
    foreach d from 0 to n-1 do Simplify(Root,d);
end
```

Notice: In line 12 we guarantee the performance of the last operation Simplify on all non deleted Cells.

4.3. Algorithm correctness

We show that algorithm allows to make reductions which are homologous invariants. We observe that the homology groups can be changed only in those lines of procedure Simplify(W, d) which modify BoCells and CoCells. Moreover, let us observe that the pair (c, b) is reduced if $\langle c, b\rangle$ is reversible (in this case $\langle c, b\rangle$ equals 1, or –1). The Cell b has dimension d, c has dimension d + 1. Reduction made by Simplify influences the boundary (d + 1)-dimensional cells in the following way:

- $\partial v = \partial v - \langle c, b\rangle\langle v, b\rangle\partial c$

Because b and c are deleted, the information about them must be also deleted from the boundaries and coboundaries of other cells. It results in the following modification of the boundaries of (d + 2)-dimensional cells:

- $\partial w = \partial w - \langle w, c\rangle c$

The boundaries of the cells of different dimensions stay unchanged.

Now it suffices to use the same arguments as in [1].

5. ANALYSIS OF ALGORITHM COMPLEXITY

Symbols:

• n	–	dimension of the cubical set,
• N	–	number of the cubes in the cubical set,
• $N(d)$	–	number of d-dimensional cells in the cubical set,
• $B(N)$	–	maximal possible number of BoCells,
• $B(N, d)$	–	maximal possible number of d-dimensional BoCells,
• C(N)	–	maximal possible number of CoCells,
• $C(N, d)$	–	maximal possible number of d-dimensional CoCells,
• k	–	number of the levels in the tree (number of the levels from the root to children),
• δB	–	coboundary of cell B,
• ∂B	–	boundary of cell B.

Let us observe that $B(N, d) = C(N, d + 1)$. Moreover, we assume that δb and ∂b are sorted lists.

Let us observe that it suffices to analyze the complexity of realization of all Simplify and Concatenate procedures and the complexity of data preparation for Main procedure. It is because CreateCube procedure is performed in time $O(1)$ and processing all nodes in the tree is performed in time $O(Nk)$.

For further analysis we need the following lemmas:

Lemma 1. *Each cubical set B built of N-cubes can. be transformed in a homologically equivalent way into A cubical set so that* $A \subset [1\ldots 2N]^n$. *This operation can be performed in* $O(N \log(N))$ *time.*

Proof. We should sort the cubes due to coordinates. It can be performed in $O(N \log(N))$ time. For a moment let us assume that B is one-dimensional and it is composed of s disjoint pieces S_1, ... S_s of the length m_1 ... m_s respectively. Let us move the pieces so that the distance between them equals 1. This operation can be performed in $O(N)$ time. This way we obtain a new cubical set A. Moreover, A is homologically equivalent to B and is contained in the cube of length $(m_1 + 1) + ... + (m_s + 1) = N + S \leq 2N$. If B has higher dimension it suffices to repeat the operation for each of the coordinates, separately. □

The following conclusion can be drawn:

Conclusion 1. The height of the tree which represents B can be limited by $\log(2N)$ in $O(N \log(N))$ time.

Now we can assume that k is not bigger than $\log(2N)$.

Lemma 2. *Cost of all Concatenate operations is up to $O(C(N)\log(N))$.*

Proof. Executing Concatenate operation on a given level of tree involves processing all boundaries and coboundaries and therefore it costs $O(C(N))$. As tree's height is not higher than $\log(2N)$, therefore a cost of all Concatenate operations on all the levels in tree is up to $O(C(N)\log(N))$. □

Lemma 3. *Numbers $B(N, 0) = C(N, 1)$ and $B(N, n-1) = C(N,n)$ are of order $O(N)$.*

Proof. Observe that during Main procedure $(n-1)$-dimensional Cells have no more than two-element coboundary. From this follows that $C(N, n)$ is of order $O(N)$. Moreover, the number of 1-dimensional Cells is up to N. Each 1-dimensional Cell has no more than two-element boundary and therefore $B(N, 0)$ is of order $O(N)$. □

Lemma 4. *Cost of all Simplify$(*, 0)$ is $O(N\log^2(N))$.*

Proof. Observe that Y is a one-element list. Moreover, in line 11 CoCells from δb are copied to δy so that Index(y) > Index(b). Therefore:

A) CoCells are copied from one coboundary to the other coboundary with higher Index value.

Now we prove the following property:

B) If y is 0-dimensional Cell then Index$(y) \le \log(2^{n+1}N)$.

First, observe that cost of lines 9-13 is greater than cost of lines 5-8 (because $d = 0$ and to estimate costs of lines 9-13, 5-8 it suffices to estimate cost of processing the list X). Therefore it suffices to study only lines 9-13 in Simplify$(*, 0)$ procedure.

For arguments simplification we assume that:

- CoCells in Simplify$(*, 0)$ are not deleted. Moreover, line 11 is replaced by – add δb to δy with
- In particular, during reduction (lines 9-13) the sequence $X \cup \{c\}$ is added to δy without deletion of the same CoCells. Some CoCell may point to deleted Cells and therefore δy may contain two CoCells which point to the same Cell.

We show the following invariants of Simplify$(*, 0)$:

(1) If δy contains at least two equivalent CoCells (two CoCells which point to the same Cell), then there are exactly two of them and they point to deleted edge or edge with one-element boundary.

(2) $\#\delta y \ge 2^{\mathrm{Index}(y)}$.

The qualities (1)-(2) are fulfilled before first reduction.

We consider next reductions. If Index$(b) \ne$ Index(y) before the current reduction, then after reduction Index(b) < Index(y) and (2) is fulfilled. If Index(b) = Index(y) before the current reduction then Index(y) = Index$(b) + 1$ after reduction. Moreover, the length of δy is not smaller than $2^{\mathrm{Index}(b)} + 2^{\mathrm{Index}(b)} = 2^{\mathrm{Index}(b)+1}$, so point (2) also occurs.

For the proof of point (1) we should take x which belongs at the same time to δb and δy. In particular, $\langle x, b\rangle$ and $\langle x, y\rangle$ do not equal 0. From this follows that before reduction x has two-element boundary (this boundary contains b and y). Moreover, after reduction x is deleted or has boundary with only one element (b is deleted). From this follows that δy by cannot contain two duplicates pointing to x. This finishes the proof of (1).

Due to point (1), the length of any δy is not bigger than $2^{n+1}N$. Due to (2) we have

$$2^{\mathrm{Index}(y)} \le \#\delta y \le 2^{n+1}N.$$

We conclude the property B): Index$(y) \le \log(2^{n+1}N)$.

Now we notice that during the algorithm performance, 1-element Cell of type CoCell can be copied to the coboundary of another Cell (δy) only $\log(2^{n+1}N)$ times. It results from the properties A)-B). Because element insertion into δy costs $O(\log(N))$ therefore cost of all Simplify$(*, 0)$ reductions is equal to $O(N\log^2(N))$. □

Lemma 5. *Cost of all Simplify$(*, n-1)$ reductions is $O(N\log^2(N))$.*

Proof. Let Q_n be the length of the boundary for basic n-dimensional cube.

Let b be $(n-1)$-dimensional Cell from line 2 of Simplify$(*, n-1)$. Observe that $\#\delta b \le 2$ and X is one-element list. As the cost of element insertion into δy and ∂x is equal to $O(\log(N))$, therefore the cost of all Simplify$(*, n-1)$ reductions is not greater than cost of processing ∂x and Y lists and simultaneously adding X list to δy list. It suffices to count the cost of processing ∂x and Y lists in lines 6 and to multiply it by $O(\log(N))$.

For arguments simplification we assume that:

- BoCells in Simplify$(*, n-1)$ are not deleted. Moreover, we replace line 6 by

$$-\text{add } \partial c \text{ to } \partial x$$

- In particular, during reduction the sequence ∂c is added to ∂x without deleting the equivalent BoCells (two BoCells pointing to the same Cell). Some BoCells may point to a deleted Cell and ∂x contains duplicates.

Observe that x is n-dimensional Cell. Let us define:

$$\mathrm{MinIndex}(x) := \min_{d\in\partial x,\, d \text{ no points to deleted cell}} \{\mathrm{Index}(d)\}.$$

We prove the following invariants of Simplify$(*, n-1)$:

(1) If ∂x contains at least two equivalent BoCells (two BoCells pointing to the same Cell) then there are exactly two of them and they point to a deleted $(n-1)$-dimensional Cell.

(2) If $y \in \partial x$ then $\#\partial x \ge 2^{\mathrm{Index}(y)}$.

(3) Executing lines 6-7 increases MinIndex(x) value.

Let $c \in \delta b$. Observe that $\#\partial x \ge 2^{\mathrm{Index}(b)}$ as $b \in \partial x$. Moreover, if $y \in \partial c$ and y do not point to a deleted cell, then Index(y) $\geq$ Index(b). If Index(y) < Index(b), then y points to a cell reduced before b so it currently points to a deleted cell.

If we consider the situation before the first reduction, then the qualities are fulfilled. We consider next reductions.

For the proof of point (1) we take (n – 1)-dimensional BoCell y which belongs at the same time to ∂c and ∂x. In particular, before reduction y has two-element coboundary (this coboundary contains c and x). Moreover, after reduction y is deleted. From this follows that ∂x contains y and only one duplicate of y. This finishes the proof of (1).

First, observe that before reduction Index(z) $\geq$ Index(b) for all $z \in Y \cup \partial x$ which do not point to a deleted Cell. Moreover, $\#\partial x \geq 2^{\mathrm{Index}(z)}$ for all $z \in Y \cup \partial x$ which point to deleted Cells. If before reduction there exists $z \in Y \cup \partial x$ such that Index(z) > Index(b), then after reduction $\#\partial x \geq 2^{\mathrm{Index}(z)}$. If before reduction Index(z) = Index(b) for some $z \in Y \cup \partial x$, then after reduction Index(z) = Index(b) + 1 and

$$\#\partial x \geq 2^{\mathrm{Index}(b)} + 2^{\mathrm{Index}(b)} = 2^{\mathrm{Index}(z)}$$

after reduction, which finishes the proof of (2).

Observe that Index(z) > Index(b) for all $z \in \partial x \backslash b$ which do not point to deleted cells after reduction. Before reduction MinIndex(x) = Index(b) and after reduction MinIndex(x) > Index(b) + 1. From this follows (3).

Due to (1) follows that $\#\partial x \leq 2Q_n N$. Moreover, due to (2):

$$2^{\mathrm{MinIndex}(x)} \leq \#\partial x \leq 2Q_n N$$

and therefore MinIndex(x) $\leq$ log($2Q_nN$). Due to (3) we conclude that the single (n – 1)-dimensional element BoCell is processed in lines 6-7 (during program running) no more than MinIndex(x) $\leq$ log($2Q_nN$). Now because the number of all (n – 1)-dimensional BoCells is $O(N)$ (Lemma 3) therefore the cost of processing ∂x, Y lists in line 6 of all Simplify(*, n – 1) reductions is $O(N \log(N))$. Therefore the cost of all Simplify(*, n – 1) reductions is $O(N \log^2(N))$. □

Theorem 2. *The homology groups of the cubical sets located in* $\mathbb{R}^2$ *can be calculated in* $O(N\log^2(N))$ *time.*

Proof. Due to Lemma 1 the cost of data preparing for Main procedure is equal to $O(N\log(N))$. Moreover, due to Lemmas 2-3 the cost of all Concatenate operations is equal to $O(N\log(N))$. Now it suffices to observe that due to Lemmas 4-5 the cost of all Simplify(*, 1), Simplify(*, 0) reductions is $O(N\log^2(N))$. □

6. NUMERICAL EXPERIMENTS

In our program we create and remove many small objects with identical size (CoCell type or BoCell type). Therefore, we decided that proper structures are created at the moment of their first using. Additionally for the objects with the same size we applied a simple memory managing algorithm which is supposed to reduce the frequency of allocating memory directly from the system:

```
Algorithm 7: MemAlloc(P,T)
input  : Pointer P, Boolean T.
output: If T =False then P =0 else P points to new object of size N.
1  Let L be the static list of objects with the same size N.
2  begin
3      if T =False then
4          Add P to L;
5          P =0;
6      end
7      else
8          if L is empty then
9              Allocate from the system 1000 objects of size N and add these objects
               to L;
10         end
11         Let C points to an object from L;
12         P =C;
13         Delete C from L;
14     end
15 end
```

Let us observe that memory allocated by MemAlloc from the system is destroyed after the program has been ended. Therefore to create and destroy the same object several times it suffices to change some pointers.

For 3-dimensional cubical sets our algorithm theoretically needs the similar number of operations as in [1]; however we receive better results than in [1].

In Table 1 we present the effects of our algorithm on a cubical set equivalent with a cube. Moreover in Table 2 we present results which we obtained for 3-dimensional cubical set equivalent with $S^1 + S^2 + T^2$, (circle + sphere + torus; in this case $H_0 = \mathbb{Z}$, $H_1 = \mathbb{Z}^5, H_2 = \mathbb{Z}^2$). The experiments were carried out on Pentium 100 with 64 MB RAM, under Linux.

Table 1.

L.p.	n	k	N	Max Mem (MB)	Reduction (MB)	T (s)	[1] T (s)
1	2	5	1 024	1	1	0.91	
2	2	6	4 096	2	3	2.13	
3	2	7	16 384	5	11	6.75	
4	2	8	65 536	6	41	25.49	
5	2	9	262 144	7	163	109.10	
6	2	10	1 048 576	7	649	458.72	
7	3	9	512	1	2	1.17	3.5
8	3	12	4 096	4	10	6.82	67.0
9	3	15	32 768	10	77	54.55	1 998.0
10	3	18	262 144	22	617	485.74	78 269.0

Table 2.

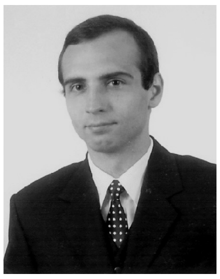
L.p.	n	k	N	Max Mem (MB)	Reduction (MB)	T (s)	[1] T (s)
1	3	12	4 020	4	10	6.68	66.4
2	3	15	32 644	10	77	53.78	1903.4
3	3	18	261 924	21	614	484.16	

In all the tables n – means dimension, k – the number of levels in binary tree which is representing a given cubical set, N – the number of elementary cubes in a cubical set, Max Mem – the maximal size of the program with data during running, Reduction – the total size of allocated and deallocated memory during running, $T(s)$ – the time of calculations, [1] $T(s)$ – the time of calculations from [1] (calculations from [1] were performed on Sun SPARC-Station 20 machine with 160 MB RAM).

References

[1] W. D Kalies, K. Mischaikow and G. Watson, *Cubical approximation and computation of homology*, Banach Center Publ. **47**, 115-131 (1999).

[2] T. Kaczynski, K. Mischaikow and M. Mrozek, Computational Homology, Springer-Verlag New York 2004.



Piotr Kot. I graduated in Mathematics from Jagiellonian University in 1995. My M.A. thesis was entitled "Contractive holomorphic metrics" and was prepared with the assistance of prof. M. Jarnicki. In 1998 I graduated in Computer Science from Jagiellonian University. My M.A. thesis was entitled "Homology calculation of cubical complexes in $\mathbb{R}^n$". It was prepared with the assistance of dr. M. Ślusarek. The same year I started to work as a junior assistant at Krakow University of Technology. In 2002/2003 I prepared PhD dissertation entitled "Exceptional sets" with the assistance of associate prof. P. Jakóbczak. This thesis was a result of over two-year research into the behavior of holomorphic functions integrals along complex directions and provided many new results. I received PhD in November 2003. Recently my research is connected with so called Radon Inversion Problem which has some applications in Computer Tomography.