Computational Homology of cubical and permutahedral complexes

by

Fintan Hegarty

School of Mathematics, Statistics, and Applied Mathematics
Supervised by Prof. Graham Ellis

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

AT

NATIONAL UNIVERSITY OF IRELAND, GALWAY

UNIVERSITY ROAD, GALWAY, IRELAND

September 2012

Abstract

Homology is the study of connectivity and “holes” in spaces. The aim of this thesis is to introduce and develop a theory of permutahedral complexes for computations of homology of large data sets and to compare, using efficient implementations, the performance of this theory with that of cubical complexes. We develop practical tools, to be submitted as a GAP package, for computing the homology of n-dimensional Euclidean data sets, where $n = 1, 2, 3, 4$, and certain higher-dimensional data sets. Certain computational advantages of using a permutahedral structure are identified; the notion of a *pure B-complex* is introduced as a data type for implementing a general class of regular CW-spaces; *zig-zag homotopy retractions* are introduced as an initial procedure for reducing the number of cells of low-dimensional pure B-complexes with discrete vector field techniques being applied for further reduction; a persistent homology approach to feature recognition in low-dimensional images is illustrated; implementing these algorithms in the GAP system for computational algebra allows for their output to benefit from the system’s library of efficient algebraic procedures.

Acknowledgements

I wish to express my deepest gratitude to my supervisor, Professor Graham Ellis, for his patience and encouragement, and for his generosity both with his time and his advice. His knowledge and insight during the four years proved invaluable. Thanks are due also to Science Foundation Ireland for paying me to do something I really enjoyed. I am grateful to Dr. Christoph Kleefeld for talking to me about his work, and discussion of some potential applications of our software.

I hugely appreciate the efforts of Dr. Pádraig Ó Catháin and Mark Jennings who proofread the thesis and offered their advice, and Nick Geoghegan who answered so many of my questions when I was first learning to use the Linux operating system. I would also like to thank the staff and other postgraduates in the Department of Mathematics for their assistance and distraction as necessary, and the people in the Chess Society and the Societies Office for further distraction.

For everything else and more, I would like to thank my parents, without whom none of this would have been possible. It is to my father, who first taught me that mathematics can be fun, that this thesis is dedicated.

Contents

Introduction

Intuitively, homology is understood to be the study of the number and nature of the "holes" and connected components of a topological space [36]. The spaces dealt with in this thesis are all finite CW-spaces, with the main focus being on the development of some theory for homology computations using CW-spaces based on n-dimensional permutahedra.
There is growing interest in constructing efficient algorithms for calculating homological and homotopical properties of spaces. This is motivated by areas of research such as topological data analysis [6], image analysis [26] and a range of other applied topics [15][16].

GAP [20] is a freely distributed, open-source system for computational discrete algebra. It provides a programming language and a large library of functions implementing algebraic algorithms written therewith. As part of this thesis a fully documented GAP package [39] for computing the homology of CW-spaces arising as subspaces of low-dimensional Euclidean space (up to dimension 4) has been created for submission to GAP.
Here follows a brief outline of the structure of the thesis.

- Chapter 1 recalls the essential basic concepts used throughout the thesis and contains no new material.

- Chapter 2 introduces the notions of *tessellated space* and *pure B-complex* - specific cases of which are dealt with in later chapters.

- Chapter 3 explains what is meant by a *cubical complex* and a *cubical space* in this thesis, and examines the theory behind a selection of computations using these.

- Chapter 4 introduces and develops some theory for *permutahedral complexes* and *permutahedral spaces* analogous to that for the cubical cases in Chapter 3, and examines why certain computations using permutahedral complexes might be more efficient than those using cubical complexes.

- Chapter 5 contains an implementation of the computations for both cubical and permutahedral complexes, and compares the efficiency of these implementations.

- Chapter 6 analyses how the developed package might be applied to real-world data.

We describe an approach to computing homotopical and homological properties of finite regular CW-spaces involving a large number of cells, paying particular attention to those arising as a union of closures of n-cells with isomorphic face posets. The approach underlies the software package HAP [13] for group cohomology and related homotopical algebra, and, in the cubical case, the CAPD homology software [30][24][26]. The approach complements the PLEX [10] and KENZO [35] software packages which are aimed at the computation of the persistent homology of simplicial complexes and the homology and homotopy groups of simply connected spaces respectively.

An excellent treatment of computational topology has already been given in the textbook [26]. The approach described in this thesis is essentially the same, but our account focuses on some new computational features that may be of general interest. In particular:

(i) We introduce the notion of *pure B-complex* as a data type for implementing a range of algorithms on a general class of regular CW-spaces. The class includes certain pure cubical and pure permutahedral subspaces of $\mathbb{R}^n$.

(ii) We identify two computational advantages of permutahedral cells over cubical cells. The first is that an n-cell in the standard permutahedral tessellation of $\mathbb{R}^n$ has fewer neighbours than an n-cell in the standard cubical tessellation. This allows us to extend a practical procedure for obtaining 'minimal' homotopy retracts of cubical lattice spaces of dimension ≤ 3 to the permutahedral lattice spaces of dimension $n \leq 4$. The second advantage is that our permutahedral lattice spaces are manifolds, and thus behave nicely with respect to taking complements.

(iii) We describe a persistent homology approach to feature recognition in low-dimensional digital images.

(iv) We apply *zig-zag homotopy retractions* as an initial procedure for reducing the number of cells of low-dimensional pure B-complexes. More standard discrete vector field techniques are applied for further cellular reduction (cf. [24]).

(v) We implement our algorithms in the GAP system for computational algebra. The output of our algorithms (which takes the form of finitely presented groups, abelian groups etc.) can thus benefit from GAP's vast library of efficient procedures for symbolic algebra. We also benefit from GAP's sophisticated method selection mechanism for selecting appropriated algorithms when computing topological properties for a range of high level data types.

As motivation we suggest a number of potential applications in Chapter 6, after developing, throughout the thesis, the theory and methods which could be used.

We identify useful implementations at three levels of generality; finite regular CW-spaces, tessellated spaces, and spaces based, for example, on cubical and permutahedral lattices.

Along with computations of integral homology and persistent Betti numbers, in Chapter 6 we illustrate the homotopical nature of the approach by including an example of the computations of fundamental groups.

1 Preliminaries

This section recalls the basic definitions and concepts needed in the thesis. The material is standard and can be found, for instance, in Hatcher's "Algebraic Topology" [25] or Munkres' "Elements of Algebraic Topology" [33].

1.1 Simplicial complexes and spaces

Definition 1.1. A *finite simplicial complex* K consists of a finite set S of vertices and a collection C of non-empty subsets of S such that if $p \in C$ and $q \subset p$ then $q \in C$.

Definition 1.2. A *pure simplicial complex* K is a simplicial complex such that the inclusion maximal subsets in C all have the same dimension.

Example 1.3. The simplicial complex K where $S = \{1, 2, 3, 4\}$ and $C = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$ is a *pure simplicial complex* as all inclusion maximal subsets are of length 2. If, however, $C = \{\{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$ then C has inclusion maximal subsets of lengths 1 and 2, so K is not pure.

Definition 1.4. A subset T of $\mathbb{R}^n$ is said to be *convex* if for every pair of points within T, every point on the straight line segment that joins them is also within T.

Definition 1.5. The *convex hull* of a set of points T in $\mathbb{R}^n$ is the intersection of all convex sets containing T.

Definition 1.6. A set of points $\{s_1, \ldots, s_{n+1}\} \subseteq \mathbb{R}^n$ is *affinely independent* if the set $\{s_2 - s_1, s_3 - s_1, \ldots, s_{n+1} - s_1\}$ is linearly independent.

Definition 1.7. An *n-simplex* is the convex hull of a subset of $n+1$ affinely independent points in Euclidean space.

Definition 1.8. Let $\underline{b_1} = \{u_1, \ldots, u_n\}$ and $\underline{b_2} = \{v_1, \ldots, v_n\}$ be two ordered bases for a vector space V. Let $A : V \to V$ be the matrix representing the identity map with respect to the bases $\underline{b_1}$ and $\underline{b_2}$. The bases $\underline{b_1}$ and $\underline{b_2}$ are said to have the same orientation if A has positive determinant, otherwise they have opposite orientation. If V is non-zero there are precisely two equivalence classes determined by the equivalence relation which is defined for all bases of V which have the same orientation. An *orientation* on V is an assignment of $+1$ to one equivalence class and -1 to the other.

We define the *closed n-ball*

$$E^n = \{x \in \mathbb{R}^n : ||x|| \leq 1\},$$

the *n-sphere*

$$S^n = \{x \in \mathbb{R}^{n+1} : ||x|| = 1\}$$

and the *open n-ball*

$$D^n = \{x \in \mathbb{R}^n : ||x|| < 1\}.$$

Definition 1.9. An $(n+1)$-dimensional topological space X is *Hausdorff* if any two distinct points s_1, s_2 of X can be separated by open sets d_1, d_2 such that $s_1 \in d_1, s_2 \in d_2, d_1 \cap d_2 = \emptyset$.

Let X be a Hausdorff space with subspace $Z \subset X$. A space Y is said to be obtained from X by *attaching an n-cell* to Z if $X \subset Y$ and there exists a continuous map ϕ from the closed n-ball to Y that restricts to a continuous map ϕ' from the $(n-1)$-sphere to Z and that maps the interior of the closed n-ball homeomorphically into Y.

Definition 1.10. [12] An n-dimensional finite *CW-space* is a Hausdorff space X possessing a chain of subspaces

$$X^0 \subset X^1 \subset X^2 \subset \ldots \subset X^n = X$$

where X^0 is a discrete finite set and where, for each $d \geq 1$, there is a chain of subspaces

$$X^{d-1} = X_0^d \subset X_1^d \subset \ldots \subset X_t^d = X^d$$

with X_i^d obtained from X_{i-1}^d by attaching a d-cell to X^{d-1}.

Definition 1.11. A CW-space is said to be *regular* if every cell is attached by a map which restricts to a homeomorphism on the boundary of the cell.

Definition 1.12. The *n-skeleton* of a CW-space X is the union of the k-cells of X where $k \leq n$, and we denote by e_j^k the jth cell of the dimension k.

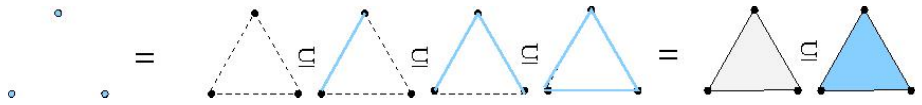


Figure 1.1: A chain of subspaces of a 2-simplex.

Definition 1.13. A *finite simplicial space* is a CW-subspace of the n-simplex.

Every finite simplicial space X can be represented combinatorially by a finite simplicial complex K, with this denoted by $X = |K|$. We say that the space X is pure if the complex K is pure.

Example 1.14. A simplicial space X consisting of a solid triangle can be represented by a simplicial complex with its set of subsets comprising one 2-dimensional cell, three 1-dimensional cells (corresponding to the edges), and three 0-dimensional cells (corresponding to the vertices).

1.2 Homology and other invariants

First, we recall some basic definitions from group theory, which can be found, for instance, in Hall's "Theory of Groups" [23]. Let G be a group and H be a subgroup. We define a *right coset* of H in G as follows: $Hg = \{hg \mid h \in H\}$. Similarly, we define a *left coset* of H in G as $gH = \{gh \mid h \in H\}$. The left (or right) cosets of H in G form a partition of G. H is *normal* if and only if the left and right cosets coincide, ie. for any $g_1 \in G$ there exists $g_2 \in G$ such that $g_1 H = H g_2$. If H is a normal subgroup of G then we can define multiplication on the cosets of H as:

$$Hg_1 Hg_2 = Hg_1 g_2.$$

Under this operation, the set of cosets of H form a group, the *quotient group G/H*.

A homomorphism of groups, $\phi : G_1 \to G_2$ is a structure-preserving map between groups. An isomorphism is an invertible homomorphism. The kernel of ϕ, $\mathrm{Ker}(\phi)$, is a normal subgroup N of G_1 and the image of ϕ, $\mathrm{Im}(\phi)$ is isomorphic to G/N.

A group G is *abelian* if $g_1 g_2 = g_2 g_1$ for all $g_1, g_2 \in G$. A *free* abelian group is one with a basis. A group G is *finitely generated* if its generating set, a set of group elements not contained in any subgroup of G other than the entire group itself, is finite. A finitely generated abelian group is *torsion-free* if for all $n \in \mathbb{N}$ the only solution of $g^n = 1$ in G is the identity. A torsion-free finitely generated abelian group is free. [37]

The *direct product* $\prod_{i \in I} G_i$ is the cartesian product of sets endowed with the natural product. A finitely generated free abelian group G is isomorphic to a direct product of copies of $\mathbb{Z}$. The number of copies is called the *rank* of G. A sequence of groups is a set of groups G_i with homomorphisms $\phi_i : G_i \to G_{i-1}$ with the property that the $\mathrm{Im}(\phi_i) \leq \mathrm{Ker}(\phi_{i-1})$. A sequence of groups is *exact* if $\mathrm{Im}(\phi_i) = \mathrm{Ker}(\phi_{i-1})$ for all i.

Definition 1.15. In this thesis a *chain complex* [5] C_* is a sequence of finitely generated free abelian groups $C_0, C_1, \ldots$ together with homomorphisms $\delta_n : C_n \to C_{n-1}$ such that $\delta_{n-1} \circ \delta_n = 0$ for all $n \geq 1$.

Definition 1.16. The *chain complex $C_*(K)$ of the simplicial complex K* is constructed as follows:

Any simplex K can be written as $\{s_0, \ldots, s_n\}$, where $s_0 \leq s_1, \ldots, s_n$ in the chosen ordering. Then $C_*(K)$ is the sequence of free abelian groups

$$\ldots \longrightarrow C_n(K) \xrightarrow{\delta_n} C_{n-1}(K) \xrightarrow{\delta_{n-1}} C_{n-2}(K) \to \ldots \to C_2(K) \xrightarrow{\delta_2} C_1(K) \xrightarrow{\delta_1} C_0(K)$$

where $C_n(K)$ is the free abelian group on the subsets of length $n+1$ in K, and the boundary homomorphism $\delta_n : C_n(K) \to C_{n-1}(K)$ is defined by

$$\delta_n(\{s_0, \dots, s_n\}) = \sum_{i=0}^{n} (-1)^i \{s_0, \dots, s_{i-1}, s_{i+1}, \dots, s_n\}.$$

The *chain complex $C_*(X)$ of the simplicial space X*, where $X = |K|$ is defined to be that of its associated simplicial complex K.

$$C_*(X) = C_*(|K|).$$

Example 1.17. Let K be the simplicial complex with $S = \{a, b, c, d\}$ and $C = \{\{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$.

Then

$$C_*(K) = 0 \to \mathbb{Z} \xrightarrow{\delta_2} \mathbb{Z}^3 \xrightarrow{\delta_1} \mathbb{Z}^3 \to 0$$

where

$$\delta_2(\{a, b, c\}) = \{b, c\} - \{a, c\} + \{a, b\},$$

$$\delta_1(\{a, b\}) = \{b\} - \{a\},$$

$$\delta_1(\{b, c\}) = \{c\} - \{b\},$$

$$\delta_1(\{a, c\}) = \{c\} - \{a\}.$$

Then

$$\delta_1 \circ \delta_2 = \delta_1(\{b, c\}) - \delta_1(\{a, c\}) + \delta_1(\{a, b\}) = 0.$$

Let $C_*(X)$ be a chain complex of a simplicial space X. The boundary homomorphism $\delta_n : C_n(X) \to C_{n-1}(X)$ has left-acting matrix representation Δ_n with respect to the canonical basis and δ_0 is taken to be the zero homomorphism. The entries in the boundary matrix Δ_n are either 0,1, or -1, with 0 indicating that the $(n-1)$-simplex is not present in the boundary of the n-simplex, and, otherwise, the sign representing the orientation of the $(n-1)$-simplex which is present in the n-simplex. Then we can calculate

- $\dim(\ker(\delta_n)) = \alpha_n - \operatorname{rank}(\Delta_n)$
- $\dim(\operatorname{image}(\delta_n)) = \operatorname{rank}(\Delta_n)$

where $rank(\Delta_n)$ is the dimension of the vector space spanned by the rows of Δ_n and α_n is the number of columns of Δ_n. In this thesis $dim(A)$ is used to mean the rank of a largest free abelian subgroup of A.
The n^{th} homology of X can now be defined.

Definition 1.18.

$$H_n(C_*(X)) = \begin{cases} \dfrac{\ker(\delta_n)}{\operatorname{image}(\delta_{n+1})} & \text{if} \quad n > 0 \\ \\ \dfrac{C_0}{\operatorname{image}(\delta_1)} & \text{if} \quad n = 0 \end{cases}$$

Henceforth $H_n(C_*(X))$, the n^{th} homology of the chain complex of X, may be abbreviated to $H_n(X)$.

Definition 1.19. The n^{th} *Betti number* of X can de defined as:

$$\beta_n(X) = \dim(H_n(C_*(X))) = \dim(\ker(\delta_n)) - \dim(\operatorname{image}(\delta_{n+1})).$$

We can think of $\beta_i(X)$ as the number of holes in X whose boundaries are made up of i-dimensional cells, with β_0 representing the number of path components.
Two points s and t in a CW-space X are said to be in the same *path component* if there is a continuous map $\rho : [0,1] \to X$ with $\rho(0) = s$ and $\rho(1) = t$.

Definition 1.20. The *Euler Characteristic* $\chi(X)$ of a simplicial space X is defined to be the alternating sum of its Betti numbers.

$$\chi(X) = \sum_{i=0}^{\infty} (-1)^i \beta_i(X).$$

One motivation for calculating homological invariants such as the Euler Characteristic and Betti numbers and homology of spaces is in order to compare and contrast spaces with one another, and analyse the effects of perturbations on a space.

Example 1.21. Consider the 2-simplex with vertices X in Figure 1.2.

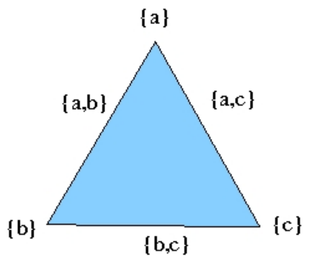


Figure 1.2: The 2-simplex X with vertices $\{a\}, \{b\}, \{c\}$.

The corresponding simplicial complex K has the collection of subsets

$$\{\{a, b, c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a\}, \{b\}, \{c\}\}.$$

The chain complex of this simplicial complex is

$$\ldots \longrightarrow \mathbb{Z} \xrightarrow{\delta_2} \mathbb{Z}^3 \xrightarrow{\delta_1} \mathbb{Z}^3 \to 0$$

where

$$\delta_2 = \begin{pmatrix} 1 & -1 & 1 \end{pmatrix},$$

$$\delta_1 = \begin{pmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 1 \end{pmatrix}.$$

Then

$$H_n(X) = \begin{cases} \mathbb{Z} & \text{if} \quad n = 0, \\ 0 & \text{if} \quad n \geq 0, \end{cases}$$

$$\beta_n(X) = \begin{cases} 1 & \text{if} \quad n = 0, \\ 0 & \text{if} \quad n \geq 0, \end{cases}$$

$$\chi(X) = 1.$$

Definition 1.22. Let C_* and C'_* be two chain complexes. A *chain map* $\Phi_* : C_* \to C'_*$ is a collection of linear homomorphisms $\Phi_n : C_n \to C'_n$ such that the diagram below

$$\begin{tikzcd}
C_{n+1} \arrow[r, "\Phi_{n+1}"] \arrow[d, "\delta_{n+1}"] & C'_{n+1} \arrow[d, "\delta'_{n+1}"] \\
C_n \arrow[r, "\Phi_n"] & C'_n
\end{tikzcd}$$

commutes for all $n \geq 0$. That is, $\Phi_n \delta_{n+1} = \delta'_{n+1}\Phi_{n+1}$ for $n \geq 0$. [12]

Since the commutativity equations $\Phi_n \delta_{n+1} = \delta'_{n+1}\Phi_{n+1}$ imply that

1. $\Phi_n(\ker(\delta_n)) \subseteq \ker(\delta'_n)$ and
2. $\Phi_n(\text{image}(\delta_{n+1})) \subseteq \text{image}(\delta'_{n+1})$

we can say that the chain map $\Phi_* : C_* \to C'_*$ induces homomorphisms

$$H_n(\Phi_*) : H_n(C_*) \to H_n(C'_*),$$

$$v + \text{image}(\delta_{n+1}) \mapsto \Phi_n(v) + \text{image}(\delta'_{n+1})$$

for all $n \geq 0$.

Example 1.23. If a simplicial complex K has a simplicial subcomplex L, then the inclusion $L \hookrightarrow K$ induces a chain map $C_*(L) \to C_*(K)$.

The definitions 1.16, 1.18, 1.19, 1.20 for simplicial spaces extend in a routine fashion to regular CW-spaces. So, too, does Example 1.23.

1.3 Homotopy

Definition 1.24. Two continuous functions f and g from a space X to a space Y are said to be *homotopic* ($f \simeq g$) if there exists a continuous function H from $X \times [0,1] \longmapsto Y$ such that $H(x,0) = f(x)$ and $H(x,1) = g(x)$. [3]

Definition 1.25. Two topological spaces X and Y are *homotopy equivalent* if there exist continuous functions $f : X \to Y$ and $g : Y \to X$ such that $f \circ g : Y \to Y$ and $g \circ f : X \to X$ are homotopic to the corresponding identities: Id_Y and Id_X. [3]

Definition 1.26. A *retracting homotopy* of a space X with subspace Y consists of a continuous function $f : X \to Y$ and the inclusion $g : Y \hookrightarrow X$ such that $f \circ g : Y \to Y$ and $g \circ f : X \to X$ are homotopic to the corresponding identities: Id_Y and Id_X. [3]

Definition 1.27. Topological spaces which are homotopy equivalent to a point are said to be *contractible.*

The main theorem underlying this thesis is the following [25]:

Theorem 1.28. *Let $f : X \to Y$ be a homotopy equivalence between CW-spaces X, Y. This map induces an isomorphism*

$$H_n(X) \cong H_n(Y)$$

for all $n \geq 0$.

Definition 1.29. A *cover* Ω of a topological space X is a collection of subsets of X whose union is the whole space X. Ω is an *open cover* if each of its members is an open set.

Definition 1.30. For some topological space X and an open cover Ω thereof, the *Cěch complex* $K(\Omega)$ is the simplicial complex constructed as follows:

- There is one vertex for each element of Ω.
- There is one edge for each pair $\omega_1, \omega_2 \in \Omega$ such that $\omega_1 \cap \omega_2 \neq \emptyset$.
- More generally, there is one k-simplex for each $k+1$-element subset $\{\omega_1, \ldots, \omega_{k+1}\}$ of Ω for which $\omega_1 \cap \ldots \cap \omega_{k+1} \neq \emptyset$.

The following theorem [25] will be useful.

Theorem 1.31. *Let X be a CW-space. Let Ω be a collection of contractible open subsets of X whose union is X. Suppose that any intersection of sets in Ω is either empty or contractible. Then there is an isomorphism*

$$H_n(C_*(X)) \cong H_n(C_*(K(\Omega)).$$

2 Tessellated Spaces

2.1 CW-spaces

A good introduction to CW-spaces can be found in Massey's "A Basic Course in Algebraic Topology" [27].We restrict our attention to finite regular CW-spaces because their cell structure can be easily stored on a computer as a binary array with each entry indicating the presence or absence of a certain facet. Since there are only finitely many cells, we can assign them a numbering and then each space can be represented as a sequence of lists $M^0, M^1, \ldots, M^n$ where the ith term of the list $M^j = \{m_1^j, m_2^j, \ldots\}$ records the $(j-1)$-dimensional cells which lie in the boundary of the ith j-dimensional cell. We also, correspondingly, encode the list of $(j+1)$-dimensional cells whose boundaries contain the ith j-dimensional cell. This can prove to be of some benefit in the implementation of certain algorithms. The j^{th} cell of dimension k in the k-skeleton of X, we denote as e_j^k.

We define the *closure* of a cell e_j^k in X to be the smallest CW-subspace $\overline{e_j^k}$ in X containing e_j^k, and the *closure* of a subset $Y \subset X$ to be the smallest CW-subspace $\overline{Y}$ in X containing Y. The sequence $R^0, ..., R^n$ is just an encoding of the partial order on the set of cells e_j^k in X given by setting $e_j^k < e_{j'}^{k'}$ if $k < k'$ and e_j^k lies in the closure of $e_{j'}^{k'}$. This *face poset* determines the regular CW-space X up to homeomorphism. We say some e_j^k is *maximal* if it does not lie in the closure of any $(k+1)$-dimensional cell. We define the *complement* of a maximal cell e_j^k in X to be the CW-space $\overline{X \backslash \overline{e_j^k}}$ arising as the closure of $X \backslash \overline{e_j^k}$.

Definition 2.1. We define the *contact complex* of a maximal cell e_j^k in X to be the CW-space $Cont(e_j^k) = \overline{e_j^k} \cap \overline{X \backslash \overline{e_j^k}}$ arising as the intersection of the closure and complement of e_j^k.

Definition 2.2. [8] We say a maximal cell e_j^k is *redundant* if $Cont(e_j^k)$ is contractible.

Definition 2.2 is motivated by J.H.C. Whitehead's theory of *simple homotopy*, an account of which can be found in [8].In this theory a CW-space Y is said to be obtained from a CW-space X by an *elementary collapse*, $X \searrow Y$, if

1. Y is a CW-subspace of X,
2. $X = Y \cup e^n \cup e^{n-1}$ with e^n, e^{n-1} cells not in Y,
3. there exists a standard ball pair (D^n, D^{n-1}), homeomorphic to $(\mathbb{I}^{n-1} \times \mathbb{I}, \mathbb{I}^{n-1} \times 1)$ with $\mathbb{I} = [0,1]$, and a characteristic map $\phi\colon D^n \to X$ for e^n which maps D^{n-1} to Y^{n-1},
4. the restriction of ϕ to the closure $\overline{\partial D^n \setminus D^{n-1}}$ of the boundary of D^n minus D^{n-1} is a characteristic map for e^{n-1} .

Proposition 2.3. *If e^k_j is a redundant maximal cell in the finite regular CW-space X then the inclusion map $\overline{X \backslash \overline{e^k_j}} \hookrightarrow X$ is a homotopy equivalence, and $\overline{X \backslash \overline{e^k_j}}$ is a homotopy retract of X.*

For some regular CW-spaces it is easy to design an efficient algorithm to recognise particular redundant cells therein. We shall call such an algorithm a *redundancy test.* Such a test is said to be *optimal* if it will recognise every redundant cell. Proposition 2.3 yields the following two algorithms.

Algorithm 2.4. *Input:* A finite regular CW-space X, a CW-subspace $A \subset X$, and a redundancy test for X.
Output: A CW-subspace X' which is a homotopy retract of X such that $A \subset X' \subset X$. If the redundancy test is optimal X' will be minimal.
Procedure:

```
Initialise $X' := X$
while there is some cell $e^k_j$ in $X'$ not in $A$ which is recognisably a redundant
cell of $\overline{X' \backslash \overline{e^k_j}}$ do
    Set $X' := \overline{X' \backslash \overline{e^k_j}}$.
end while
```

Algorithm 2.5. *Input:* A finite regular CW-space X, a non-empty CW-subspace $A \subset X$, and a redundancy test for X.

Output: A CW-subspace X' such that A is a homotopy retract of X' and $A \subset X' \subset X$. If the redundancy test is optimal X' is minimal.

Procedure:

```
Initialise $X' := A$
**while** there is some cell $e^k_j$ in $X$ which is recognisably a redundant cell of $X' \cup \overline{e^k_j}$
**do**
  Set $X' := X' \cup \overline{e^k_j}$.
**end while**
```

The *cellular chain complex*

$$C_*(X): \ \ldots \to C_k(X) \xrightarrow{\delta_k} C_{k-1}(X) \to \ldots \to C_0(X)$$

of a finite regular CW-space X is constructed by taking $C_k(X)$ to be the free abelian group with free generators corresponding to the k-cells of X, and with the boundary homomorphisms given by

$$\delta_k(e^k_j) = \sum \epsilon^k_{ij} b^k_{ij} e^{k-1}_j$$

where

$$b^k_{ij} = \begin{cases} 1 & \text{if } e^{k-1}_i \text{ lies in the closure of } e^k_j \\ 0 & \text{otherwise} \end{cases}$$

$$\epsilon^k_{ij} = \begin{cases} 0 & \text{when } b^k_{ij} = 0, \\ 1 & \text{when } k = 1, b^1_{ij} = 1, b^1_{i'j} = 1, i < i', \\ -1 & \text{when } k = 1, b^1_{ij} = 1, b^1_{i'j} = 1, i > i', \\ \pm 1 & \text{when } k > 1, b^k_{ij} = 1, \text{ with sign uniquely determined by the equation } \delta_{k-1}\delta_k = 0 \end{cases}$$

Definition 2.6. Let X be a regular CW-space with CW-subspace $Y \subset X$. From the induced injective chain map

$$\Phi_x : C_*(Y) \to C_*(X)$$

we define the quotient chain complex:

$$C_*(X/Y) = C_*(X)/C_*(Y),$$

where $C_n(X/Y)$ is the quotient free abelian group $C_n(X)/C_n(Y)$ and the boundary on $C_*(X)$ induces the boundary homomorphism $C_n(X/Y) \to C_{n-1}(X/Y)$.

Definition 2.7. From the above definition, given a regular CW-space X and subspace $Y \subset X$, we define $H_n(X, Y) = H_n(C_*(X/Y))$.

The following is a standard result [25]:

Theorem 2.8. *If X is a space and Y is a nonempty closed subspace which is a retract of some neighbourhood in X then there is an exact sequence*

$$\ldots \to H_n(Y) \xrightarrow{i} H_n(X) \xrightarrow{j} H_n(X, Y) \to H_{n-1}(Y) \to$$

for $n \geq 1$ where i is the inclusion $Y \hookrightarrow X$ and j is the quotient map $X \to X/Y$.

Further, if $Z \subset Y \subset X$ and the closure of Z is contained within the interior of Y then Z can be excised*, meaning we can 'cut out' Z from Y and X, such that the relative homologies of the pairs (X, Y) and $(X/Z, Y/Z)$ are isomorphic.*

Use can be made of such excisions to reduce homology computation time considerably.

Example 2.9. The functions used in the following extract of GAP code are described in greater detail later in the thesis, but for the moment we informally define `ChainComplex` to be a function which returns the chain complex of a CW-space, `ContractibleSubcomplex` to be a function which returns a contractible subcomplex of a space and `Homology` to be a function which returns the homology of a space. `ChainComplexOfPair` is a function which inputs a space X and a subspace Y and returns the quotient $C(X)/C(Y)$ of cellular chain complexes. More technically, given a space X and a contractible subspace Y, we can calculate the homology of X by considering only those cells which are in X but not in Y. From the GAP code, then, it is clear that the computation is substantially quicker when the chain complex of the pair is used. This confirms what we would intuitively expect from looking at Figure 2.2(a). There are substantially fewer coloured pixels, upon which the necessary calculations are performed, therein than in Figure 2.1.

```
gap> B:=ChainComplex(A);; time;

236

gap> C:=ContractibleSubcomplex(A);;time;

68

gap> D:=ChainComplexOfPair(A,C);;time;

88

gap> for i in [0,1] do Homology(B,i);; od; time;

399537

gap> for i in [0,1] do Homology(D,i);; od; time;

460
```

Timings here and throughout the thesis, unless otherwise specified, are in milliseconds.

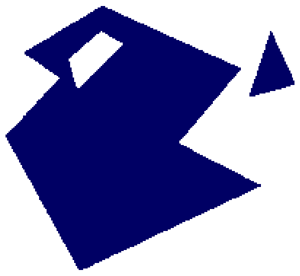
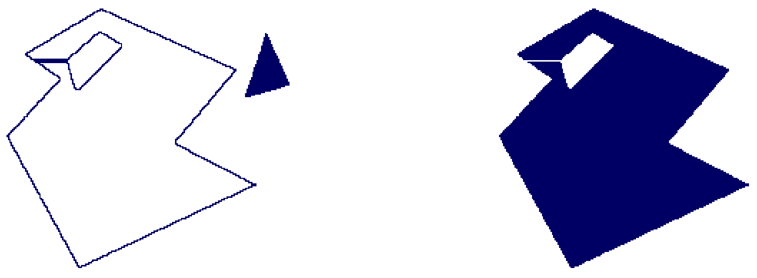


Figure 2.1: A space X with 4446 cells, where $X = |A|$.



(a) A': A with contractible subcomplex excised.

(b) $'A$: The interior subcomplex of A which was excised.

Figure 2.2: Excision of a contractible subcomplex.

2.2 Tessellated Spaces

In this section we introduce the notions of *tessellated space* and *pure B-complex* and describe an efficient representation.

Definition 2.10. In this thesis the term *tessellated space* is used to mean a regular CW-space X such that for some $n \geq 0$

- (i) X is the union of the closures of its n-cells
- (ii) all closures of n-cells have face poset isomorphic to that of some fixed polytope.

These n-cells might all be n-simplexes, or all n-cubes, or all n-permutahedra. In this thesis we focus on the latter two cases. We say that X has dimension n and that the closure of any n-dimensional cell is a *facet* of the space. Thus X is the union of its facets.

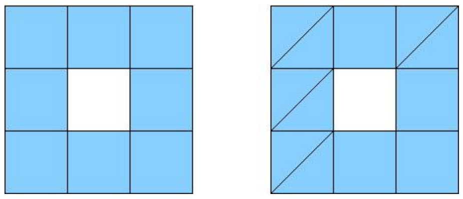


Figure 2.3: The space on the left is regularly tessellated, with all cells having a poset with four vertices, whereas the space on the right is not as some of its cells have posets with 3 vertices and others have 4.

Restrictions (i) and (ii) make it possible to efficiently represent such spaces on a computer in such a way that a quick computation yields all those facets incident with a given facet f.

Definition 2.11. Let X be a tessellated space and let e, f be two of its facets. If $e \cap f \neq \emptyset$ then e and f are *neighbours*. Note that f is a neighbour of itself.

The *neighbourhood* of a facet f in some space X is defined to be the union $N_X(f)$ of all neighbours of f. Let Y be a tessellated subspace of X (i.e. some union of facets of X). The *neighbourhood* of Y is defined to be the union $N_X(Y)$ of all the neighbourhoods $N_X(f)$ of facets f of Y. Note that $N_X(Y)$ is again a tessellated subspace of X.

A tessellated space is *finite* if it has only finitely many facets.

A finite pure simplicial complex is an example of a tessellated space and Algorithms 2.4 and 2.5 are theoretically applicable, but the representation is not efficient as the computation of the contact complex of an n-simplex would be expensive.

An analogous representation of a CW-subspace $X \subset \mathbb{I}^N$, where $\mathbb{I}$ is the unit interval, is described at the end of Chapter 3. There, X is represented as a finite collection of pairs of subsets $K = \{\tau \subset \sigma\}$ where τ, σ are subsets of the set $S = \{1, \dots, 2^N\}$ with $|\tau - \sigma| = n$ having a geometric realisation as an n-cube. As with the simplicial complex, though mathematically faithful, this representation is not computationally efficient.

The notion of a B-complex, which we explain in Definition 2.13, lends itself well to the application of the aforementioned algorithms.

Definition 2.12. A *binary array* of dimension 1 is a list $b = [x_1, x_2, \dots, x_n]$ where each x_i is either 0 or 1. The *array size* of b is defined to be the singleton $[n]$. A binary array of dimension $d > 1$ is a list $[x_1, x_2, \dots, x_n]$ where each x_i is a binary array of dimension $d - 1$ and all x_i have the same array size. The *array size* of b here is defined to be the list of integers got by appending the array size of x_i to $[n]$.

For some set of points $P = p_1, p_2, \dots$ in $\mathbb{R}^n$, we use Dirichlet-Voronoi cell to mean the region described by

$$D(p_i) = \{x \in \mathbb{R}^n : ||p_i - x|| \leq ||p_j - x|| \text{ for any } p_j \in P\}.$$

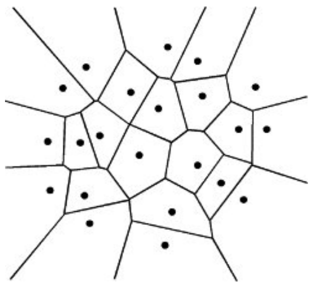


Figure 2.4: Dirichlet-Voronoi diagram for a set of points in the plane.

Let L be a *lattice* in $\mathbb{R}^n$, ie. an additive subgroup generated by n linearly independent vectors in $\mathbb{R}^n$. [9] Each point $v \in L$ determines a Dirichlet-Voronoi cell, discussed in more detail in [40], and Euclidean space $\mathbb{R}^n$ can be assigned the structure of a tessellated space whose facets are these cells, $D(v)$. This tessellated space is denoted by $\mathbb{R}^n_L$.

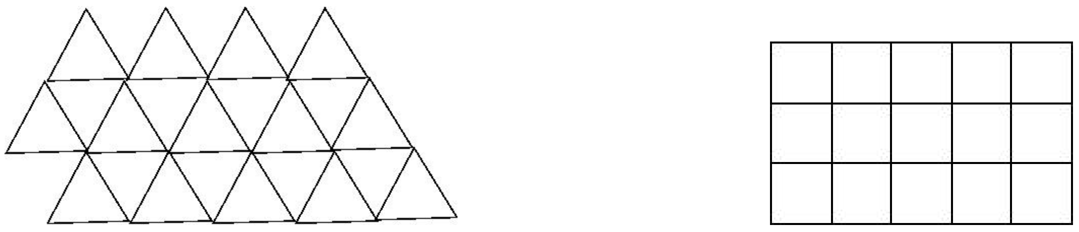


Figure 2.5: A tessellation of simplices can not be assigned a lattice structure but a cubical tessellation can, lending itself to a simple representation as a binary array

Definition 2.13. A *pure B-complex* $K = (A, B, \underline{T})$ consists of

- a binary array $A = (a_\lambda)$ of dimension d,
- a basis $\underline{T} = \{t_1, t_2, \ldots, t_d\}$ for $\mathbb{R}^d$,
- a finite set $B \subseteq \mathbb{Z}^d$. We call B the *ball*.

Definition 2.14. Given some $\lambda \in \mathbb{Z}^d$ such that $a_\lambda = 1$ we define the *neighbourhood* $N(a_\lambda)$ to be the union of $a_{\lambda+b} : b \in B$.

Every pure B-complex provides a lattice structure which corresponds to the tessellated space $X \subseteq \mathbb{R}^d$ with facets $D(\lambda_1 t_1 + \lambda_2 t_2 + \ldots + \lambda_d t_d)$ for all λ with $k_\lambda = 1$, where $D(v)$ is a Dirichlet-Voronoi cell. In Example 2.15, the basis used is $(1,0),(0,1)$, and the entry, $a_{(3,2)}$, for example, in A has value 1, giving $t_1 = (1,0), t_2 = (0,1), \lambda_1 = 3, \lambda_2 = 2$. The facet defined by the Dirichlet-Voronoi cell $D(3*(1,0)+2*(0,1))$, ie. $D(3,2)$ is therefore present in X.

Example 2.15. Here the pure B-complex K with binary array A has basis $(1,0),(0,1)$.

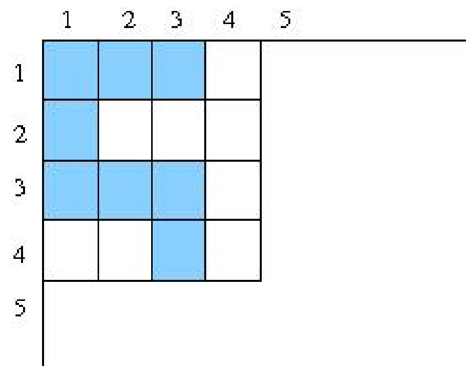


Figure 2.6: $|K| = X \subset \mathbb{R}^n$.

Figure 2.6 has binary array

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}.$$

From such a binary array representation we can easily compute the neighbourhood of any facet and thus directly compute the number of path components. The following algorithm can be used.

Algorithm 2.16. Path Components

Input: A finite tessellated space X represented in such a way that the neighbours can be easily computed.

Output: A list $X_1, \ldots, X_c$ of the path components of X.

Procedure:

```
Initialise $c := 0$
Deem all facets of $X$ to be uncoloured
**while** there exists an uncoloured facet of $X$ **do**
  Set $c := c + 1$.
  Choose some uncoloured facet $g$ and assign it the colour $c$.
  **while** there exists some uncoloured facet $f$ in the neighbourhood of a facet
  $g$ of colour $c$ **do**
    Assign the colour $c$ to $f$.
  **end while**
**end while**
For $0 \leq i \leq c$ let $X_i$ be the union of the facets of colour $i$.
```

Chapter 3 of this thesis focuses on cubically tessellated spaces which are B-complexes with orthonormal basis, Chapter 4 on permutahedrally tessellated spaces which are B-complexes with a certain other basis which is described in that chapter, and Chapter 5 deals with the computational implementation of functions for objects with either cubical or permutahedral lattice structure.

2.3 Persistent Homology

In this section we show how our methods can be used to calculate persistent homology, after Carlsson et al. [7]

Algorithm 2.17. Thickened Pure Complex

Input: A pure complex K.

Output: A pure complex L such that for any entry 1 in the pure complex K all neighbouring entries of the corresponding entries in L equal 1.

Procedure:

```
Initialise $L := K$
```

```
for $k_\lambda = 1$ do
    for $k_\kappa \in N_K(k_\lambda)$ do
        Set $l_\kappa = 1$.
    end for
end for
```

Example 2.18. Figure 2.7 shows a number of iterations of the thickening algorithm over a set of points X_1 sampled from some space and the table below shows the Betti numbers of the space after each thickening. It may be easy for a human to interpret from the graphical representation X_1 that the data is sampled from a space with a single path component and two 1-dimensional holes, which would seem to fit with the computer's result that after a number of thickenings the number of 1-dimensional holes settles down to 2 for a considerable number of iterations. The Betti numbers at the nth of these iterations are shown in Table 2.1.

Remark: Our method of thickening, using neighbourhoods, provides a certain consistency and gives us more control over the filtration than we would have if we were thickening using simplicial complexes, for example.

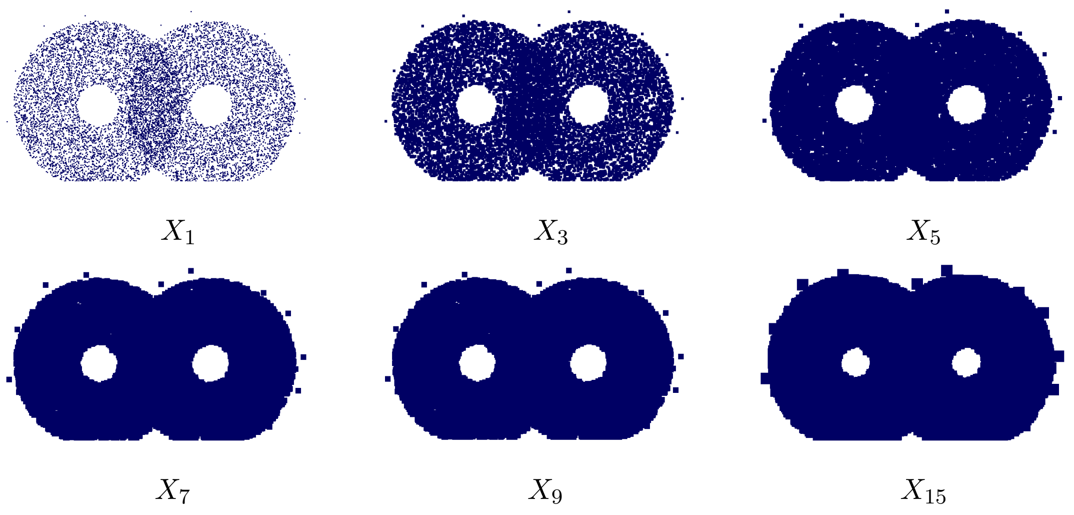


Figure 2.7: A selection of thickenings of X_1.

However, when attempting to interpret the homology type of the original space from

Table 2.1: $\beta_n(X)$ at various thickenings.

β_n	X_1	X_3	X_5	X_7	X_9	X_{15}
β_0	3019	16	11	10	8	1
β_1	175	1201	171	7	2	2

which the data were sampled the Betti numbers can be unreliable as they fail to capture a lot of the information. See Figure 2.8 for an example where one hole is created as another is destroyed, leaving the Betti number unchanged.

To identify when holes persist, are created, or are destroyed within a sequence of subsets of a space we analyse the *persistent homology* [44] of a space.

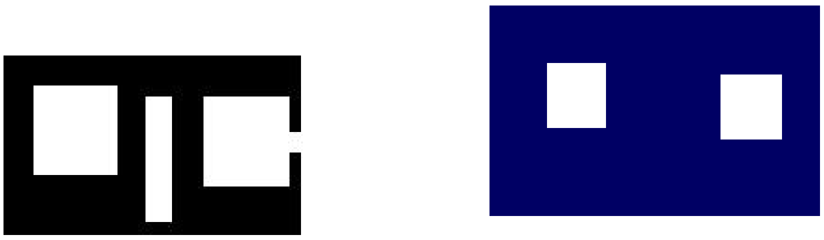


Figure 2.8: The space on the left, X has $\beta_1 = 2$. The space on the right, $\mathit{Thickened}(X)$ has $\beta_1 = 2$ also, hiding the creation and destruction of holes.

The table in Example 2.18 gives only a partial indication of homological persistence. While it shows, for example, that $H_1(X_5, \mathbb{A})$ is generated by 171 independent cycles and $H_1(X_7, \mathbb{A})$ is generated by 7 cycles, it does not indicate how many of the 171 cycles lie in the kernel of the induced homology homomorphism $H_1(X_5, \mathbb{A}) \to H_1(X_7, \mathbb{A})$, that is, it does not show whether the 7 holes are newly created or have persisted through the thickening. Let $X_1 \subset X_2 \subset X_3 \subset \ldots$ be a sequence of inclusions of finite tessellated spaces. In order to determine the *persistence Betti numbers* β_k^{ij} we

1. Compute the induced homology homomorphisms $\iota_{i,i+1} : H_k(X_i) \to H_k(X_{i+1})$;

2. Use standard linear algebra algorithms to compute the composite homomorphisms $\iota_{i,j} = \iota_{j-1,j}, \iota_{j-2,j-1}, \ldots, \iota_{i+1,i+2}, \iota_{i,i+1}$ and then

3. Determine β_k^{ij} as the rank of the homomorphisms $\iota_{i,j}$.

For an abelian group $\mathbb{A}$ any sequence of inclusions of CW-subspaces

$$X_1 \subset X_2 \subset X_3 \subset \ldots \subset X_t$$

induces a sequence of chain maps

$$C_*(X_1, \mathbb{A}) \hookrightarrow C_*(X_2, \mathbb{A}) \hookrightarrow C_*(X_3, \mathbb{A}) \hookrightarrow \ldots \hookrightarrow C_*(X_t, \mathbb{A})$$

which in turn induce a sequence of homology homomorphisms

$$H_n(X_1, \mathbb{A}) \to H_n(X_2, \mathbb{A}) \to H_n(X_3, \mathbb{A}) \to \ldots H_n(X_t, \mathbb{A})$$

for each $n \geq 0$ [12].

Definition 2.19. For a sequence of inclusions of CW-subspaces $X_1 \subset X_2 \ldots \subset X_t$ and each $n \geq 0$ we define the nth *persistence matrix* over $\mathbb{A}$ to be the $t \times t$ integer matrix $P_n = (p_{ij})$ where

$$p_{ij} = \dim(\text{image}(H_n(X_i, \mathbb{A})) \to H_n(X_j, \mathbb{A}))$$

for $i \leq j$, and

$$p_{ij} = 0$$

for $i > j$.

Following the work of Carlsson et al. [7] the matrix (p_n^{ij}) is represented by a graph called a *bar code* which has horizontal edges and vertices arranged in columns. The i^{th} column has $p_n^{ii} = \beta_n(X_i)$ vertices and there are p_n^{ij} paths from the i^{th} to the j^{th} column. The β_1 bar code for Example 2.18 is partially illustrated in Figure 2.9. Owing to the size of the bar code we have labelled each row with an integer indicating the number of repetitions thereof.
The pair of long horizontal paths in Figure 2.9 indicate that there are two 1-dimensional homology generators that persist from X_3 to X_{15}, suggesting two significant 1-dimensional

holes, which is consistent with the homology of a space comprising a single path component with two 1-dimensional holes.

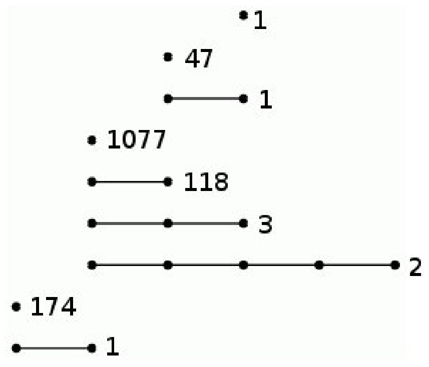


Figure 2.9: The barcode for $\beta_1(X)$ in Example 2.18

In Figure 2.10, the bar code for Figure 2.8 the destruction and creation of holes, which one could not infer from the Betti numbers alone, is clearly visible.

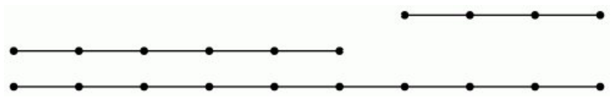


Figure 2.10: The persistent bar code indicates the creation of one hole and the destruction of another midway during the sequence for Figure 2.8.

One of the aims of this thesis is the computation of persistence Betti numbers and homology homomorphisms arising from a sequence of inclusions of tessellated spaces. The approach used here differs from the classical algorithm of Zomorodian and Carlsson [44] as we focus on tessellated spaces and make use of their special structure for our computations. From Example 2.18 one can see how the obvious approach to homology computation might run into difficulties. Consider for example the space X_9. Using the basis $(0, 1), (1, 0)$ it involves 279550 2-dimensional facets, 561088 1-dimensional edges, and 281544 0-dimensional vertices. How we obtain these is described in Chapter 3. In its chain complex

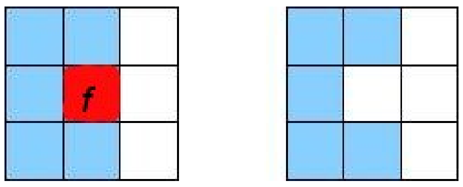
$$C_2(X_9) \xrightarrow{\delta_2} C_1(X_9) \xrightarrow{\delta_1} C_0(X_9)$$

the boundary homomorphism δ_2 is represented by a 561088×281544 matrix, and the boundary homomorphism δ_1 is represented by a 279550×561088 matrix. A direct computation of $H_1(X_9, \mathbb{Z}) = \dfrac{\ker(\delta_1)}{\text{image}(\delta_2)}$ using the Smith-Normal form algorithm would be time consuming, and liable to 'blow up' [11]. As part of this thesis we implement a number of methods which aim to efficiently compute the homology of regular tessellated spaces of up to dimension 4.

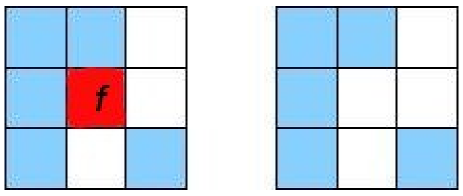
2.4 Positive Neighbourhoods

In this section we explain how our representation allows for the creation of efficient redundancy tests.

Definition 2.20. The *complementary neighbourhood* of a facet f in X is $\hat{N}_X(f) = N_X(f) \backslash f$. The complementary neighbourhood $\hat{N}_K(k_\lambda) = 1$ of an entry k_λ in the binary array of a pure B-complex K is $N_K(k_\lambda)$ but with $k_\lambda = 0$. A neighbourhood is defined to be *positive* if the complementary neighbourhood is a homotopy retract thereof.



(a) The neighbourhood on the left is positive as its homology remains the same when the central facet is removed.



(b) The neighbourhood on the left is not positive as its homology changes when the central facet is removed.

Figure 2.11: Positivity in a pure B-complex with basis $(0,1),(1,0)$.

For some lattices the number of possible combinatorial types of positive neighbourhoods is quite small and one can pre-compute a list of them. In $\mathbb{R}^n_C$ (a subspace of $\mathbb{R}^n$ with cubical tessellation - further discussed in chapter 3): 2 of the 4 possible combinatorial types in $\mathbb{R}_C$; 116 of the 256 possible combinatorial types in $\mathbb{R}^2_C$, and 41123720 of the 67108864 possible combinatorial types in $\mathbb{R}^3_C$ are positive. For higher dimensions the list becomes too big to store. The notation $\mathbb{L}^n_C$ is used to mean the list of positive neighbourhoods in n-dimensional cubical space.

This list can then be used in algorithms such as the following, which correspond to Algorithms 2.4 and 2.5 for CW-spaces in general:

Algorithm 2.21. Homotopy Equivalent Minimal Pure Subcomplex

Input: A pure B-complex K and a subcomplex $J \subset K$. $X = |K|$ and $W = |J|$.

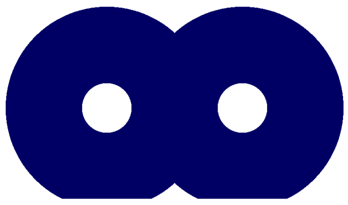
Output: A pure B-complex L, where the entry with coordinates λ is denoted by l_λ, where $Y = |L|$ such that Y is a homotopy retract of X which contains W and from which no further facets can be removed without affecting these properties.

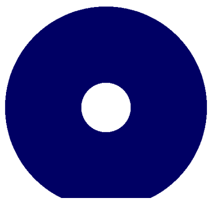
Procedure:

```
Initialise $L := K$
Let $L = L$!.binaryArray, $J =$ J!.binaryArray
while $L - J$ has an entry $(L - J)_\lambda = 1$ with $\hat{N}_L(\lambda) \in \mathbb{L}$ do
  Set $l_\lambda = 0$
end while
Return the pure $B$-complex with binary array $L$.
```

Example 2.22. The application of this algorithm can be seen in Figure 2.12.

Remark: The order in which cells are traversed can affect the returned complex, though the homotopy type will be the same for any progression. This order is not randomised, but can be changed in the implementation. In Figure 2.13(b) the first figure is the retraction of X in Figure 2.13(a) when it has been traversed from left to right then top to bottom by the retraction algorithm, whereas the second figure is the retraction when X has been traversed from right to left and bottom to top. In the third image the retraction alternates between moving left to right and top to bottom and right to left and bottom to top; switching between the two every time a positive cell is found. There is no traversal method which is guaranteed to be most efficient in all cases. The default traversal in HAP is the third of the above examples. The homotopy type of all three retractions are the same no matter which retraction is chosen.


(a) A tessellated space X.


(b) $A \subset X$.

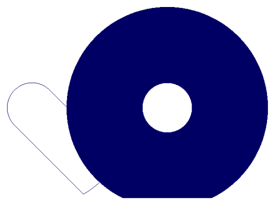

(c) The corresponding pure cubical space of the homotopy equivalent minimal pure subcomplex of X containing A.

Figure 2.12: Example of a homotopy equivalent minimal pure subcomplex.

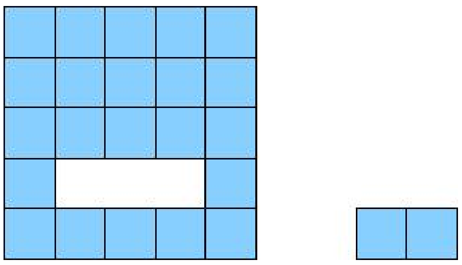
Algorithm 2.23. Homotopy Equivalent Maximal Pure Subcomplex

Input: A pure B-complex K and a subcomplex $J \subset K$. $X = |K|$ and $W = |J|$.

Output: A pure B-complex L where $Y = |L|$ such that $Y \subset X$ and contains W as a homotopy retract and to which no further facets can be added without affecting these properties.

Procedure:

```
Initialise $L := J$
Let $L := L$!.binaryArray, $J := J$!.binaryArray, $K := K$!.binaryArray
**while** $K - L$ has an entry $(K - L)_\lambda = 1$ such that $\hat{N}_K(\lambda) \in \mathbb{L}$ **do**
  Set $l_\lambda = 1$
**end while**
Return the pure $B$-complex with binary array $L$.
```


(a) The space X and $A \subset X$

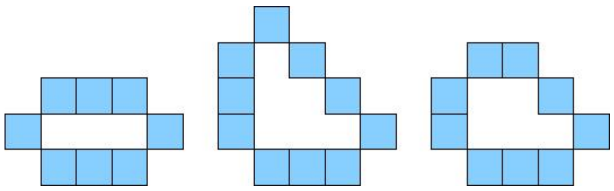
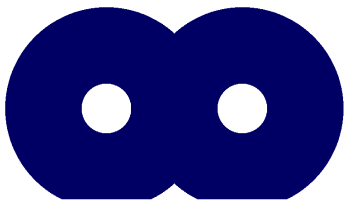
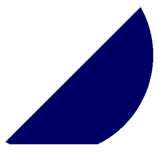

(b) Three retractions of X containing A using different traversals of X

Figure 2.13: Different orders for traversing a complex can return different retracts, but all of the same homotopy type.


(a) A tessellated space X.


(b) $Y \subset X$

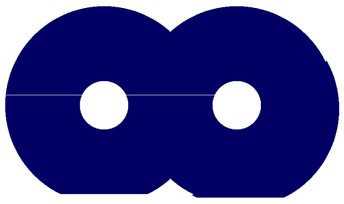

(c) A subspace of X which is homotopy equivalent to Y.

Figure 2.14: Example of homotopy equivalent maximal pure subcomplex implementation.

Example 2.24. We can see from Figure 2.14 how the resulting complex is a single path com-

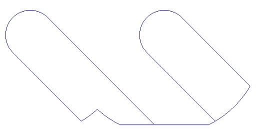


Figure 2.15: $Z = |L|$ where $L =$ `Contracted Complex`(K)

ponent with no holes. (Note: there is a line connecting the apparent 'holes' in the complex to the outside, but it may not be very visible, as the width of this path is only that of one pixel.)

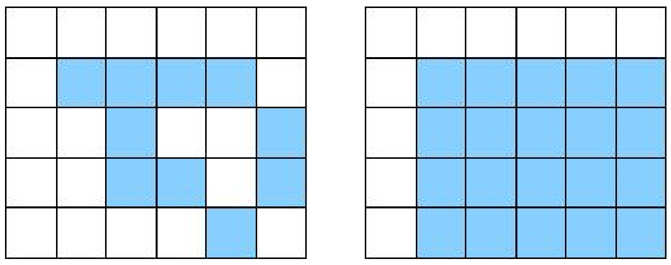
For a given pure B-complex K Algorithm 2.23 can be used to calculate a contractible subcomplex L of K. The interior of this L can then be excised as in Example 2.9, making the calculation of the relative homology significantly more efficient than simply calculating the homology of K. In Example 2.24, the chain complex of A comprised 257028 0-cells, 512358 1-cells and 255329 2-cells, whereas the relative chain complex of the pair comprised 2483 0-cells, 5339 1-cells and 2854 2-cells.

Letting the subcomplex of J for Algorithm 2.21 be the subcomplex with all zero entries allows for the retraction of any entry with positive neighbourhood in K, giving a pure complex such that no further entries can be changed to 0 without changing the homotopy type of the space associated with K. Homology computations using a contracted complex are much more efficient. See Figure 2.15 where Z is homotopy equivalent to X in Figure 2.14, but whereas K contains 255329 facets, L contains only 1396 facets.

2.5 Zig-zag retraction

In this section we introduce a new method for producing a homotopy retract of cellular spaces of low dimension.

We define a *bounding space* for X to be a contractible space containing X. To construct a bounding space for an n-dimensional space X with corresponding binary array A, we consider all $a_\lambda = 1$. Then we find the entry $a + (m_1, m_2, \ldots, m_n)$, where m_i is the minimal value of each λ_i. We also find the entry $a_{(M_1, M_2, \ldots, M_n)}$ where the M_i are the maximal values. Then every entry $a_{(p_1, p_2, \ldots, p_n)}$ in which any $m_i \leq p_i \leq M_i$ in the bounding space is assigned to be 1. See Fig. 2.16 for an example.



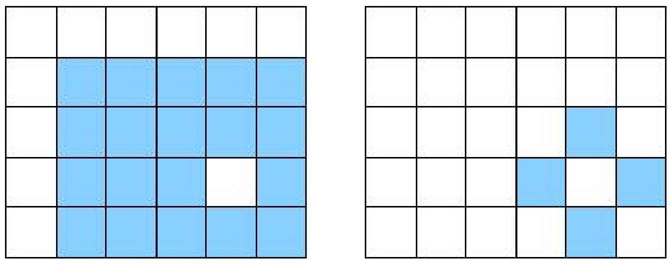
(a) X. (b) Y

Figure 2.16: Y is a bounding space for X

A *zig-zag retract* [14] of a space X is a space Y such that

$$X = B_0 \hookleftarrow A_1 \hookrightarrow B_1 \hookleftarrow A_2 \hookrightarrow B_2 \hookleftarrow \ldots \hookrightarrow B_{k-1} \hookleftarrow A_k = Y,$$

where B is some bounding space for X. A sequence of combinations of algorithms may be used to find zig-zag retracts which are smaller than actual retracts. To produce a zig-zag retract of $X = B_0$ Algorithm 2.21 can be used to produce a retract A_1. Then a small bounding space $A_1 \subset B_1'$ is constructed, and using Algorithm 2.23 a subspace $B_1 \subset B_1'$ is computed, which is maximal with respect to containing A_1 as a homtopy retract. Then Algorithm 2.21 can be used to find a retract $A_2 \subset B_1$. This process can be repeated. This leads to a smaller (or equal) retract than simply contracting a space. See Fig. 2.17 and Example 2.25.



(a) A subcomplex, W, of Y which is homotopy equivalent to X.

(b) A homotopy equivalent retract of W.

Figure 2.17: Finding a zig-zag retract of X from Fig. 2.16

Example 2.25. Given below is the binary array K, where $ZZ = |K|$ is zig-zag retract of X in Figure 2.15, as ZZ is too small to view.

$$K'' = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}.$$

Example 2.26. This extract of GAP code illustrates some difference in the output from the `ContractedComplex` and `ZigZagRetractOfPureCubicalComplex` functions. Some extra cells may be removed with zig-zag retraction, which takes slightly longer but this can lead to a significant improvement in homology computation times.

```
gap> A:=List([1..100],i->1);;

gap> A:=List([1..100],i->StructuralCopy(A));;

gap> A:=List([1..100],i->StructuralCopy(A));;

gap> A[50][50][50]:=0;;

gap> P:=PureCubicalComplex(A);;

gap> Size(P);

999999

gap> ZZ:=ZigZagContractedPureCubicalComplex(P);;time;

15652

gap> Size(ZZ);

6

gap> C:=ContractedComplex(P);;time;

8724

gap> Size(C);

6
```

In the above GAP session extract, the `ContractedComplex` is more efficient as it is faster and removes just as many cells as the `ZigZagContractedPureCubicalComplex`. Amending the complex slightly, as below, the zig-zag retraction is much more efficient, as while it takes slightly longer it removes substantially more cells, making succeeding computations on the complex simpler.

```
gap> A[70][38][29]:=0;;A[39][17][94]:=0;;

gap> P:=PureCubicalComplex(A);;

gap> Size(P);

999997

gap> ZZ:=ZigZagContractedPureCubicalComplex(P);;time;

15329

gap> Size(ZZ);

27

gap> C:=ContractedComplex(P);;time;

9221

gap> Size(C);

217

gap> F:=ChainComplex(ZZ);;time;

8

gap> G:=ChainComplex(C);;time;

6389

gap> for i in [0..3] do Print(Homology(F,i),"\n");od;time;

[ 0 ],[ ],[0, 0, 0],[ ]

400

gap> for i in [0..3] do Print(Homology(G,i),"\n");od;time;

[ 0 ],[ ],[0, 0, 0],[ ]

29990
```

2.6 Discrete Morse theory

A further algorithm for finding homotopy retracts of CW-spaces can be effectively described using discrete Morse theory [18]. A *discrete vector field* on a regular CW-space X is a collection of arrows $\alpha : s \to t$ where

1. s, t are cells and any cell is involved in at most one arrow;

2. $\dim(t)=\dim(s)+1$;

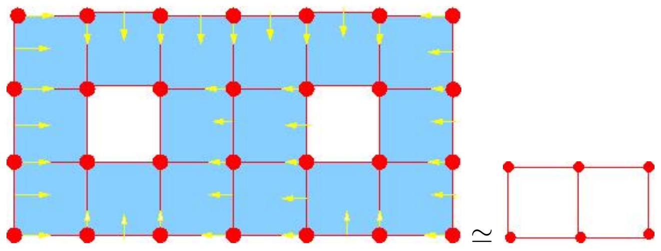


Figure 2.18: A discrete vector field on a cubical complex and a homotopy retract thereof.

3. s lies in the boundary of t.

A sequence of arrows $\alpha_1 : s_1 \to t_1, \alpha_2 : s_2 \to t_2, \ldots$ in a vector field is a *path* if

1. the s_i are cells of common dimension d and the t_i are cells of common dimension $d+1$.

2. each s_{i+1} lies in the boundary of t_i.

A discrete vector field is said to be *admissible* if it contains no infinite path of arrows and no finite cycles. The example in Figure 2.18 is admissible. An arrow $s \to t$ *involves* the cells s, t. A cell is *critical* if it is involved in no arrows of the vector field.
From [19] we take this "fundamental theorem" of discrete Morse theory:

Theorem 2.27. *If X is a regular CW-space with an admissible discrete vector field then there is a homotopy equivalence*

$$X \simeq Y$$

where Y is a CW-space whose cells are in one-to-one correspondance with the critical cells of X.

In order to create a discrete vector field on a space X, a pair (s,t) of cells is defined to be X-*free* if

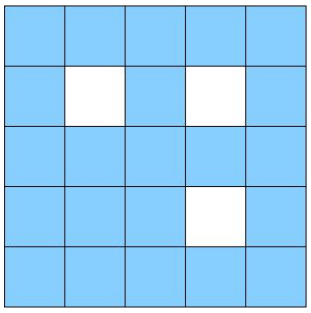
1. dim(t) =dim$(s+1)$;

2. s lies in the boundary of t but in the boundary of no other cell in X.

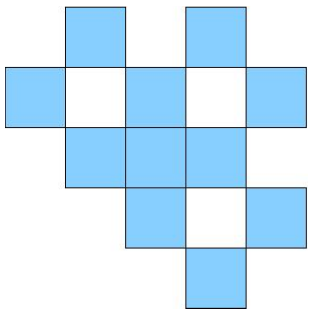
Then, to form a CW-retract of X, we assign the trivial vector field, involving no arrows, to X' which is a copy of X we shall manipulate to find a homotopy retract. While there exists an X'-*free* pair (s,t) in X', the arrow $s \to t$ is added to the vector field on X'. Eventually, this leaves an admissible discrete vector field to which no more arrows can be added, and the remaining, critical cells provide a CW-space which is homotopy equivalent to X.

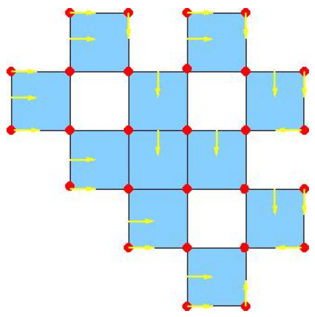
It can often be more efficient to use a combination of a tessellated retraction and discrete Morse theory than to use either on their own. Consider Example 2.28 where a cubically tessellated space is contracted efficiently, and discrete Morse theory is then used to contract the space even further, minimising computations necessary to calculate the homology.

Example 2.28. To find a retraction of X.



$|X|$


(a) First, the space is contracted using the standard contraction for pure B-complexes.


(b) A discrete vector field is then assigned.

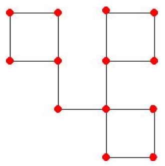
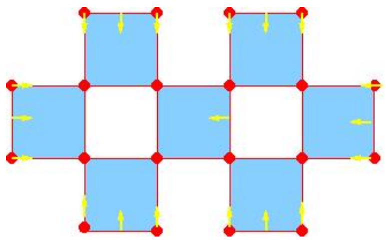

(c) Retract using Discrete Morse theory.

Figure 2.19: Contracting X, first by contracting the complex and then using Discrete Morse theory.

The chain complex for the discrete vector field is considerably smaller than the chain complex of the space which was just cubically contracted. The pure complex contraction, however, is much faster at removing large quantities of non-essential facets.

This method can also be combined with the zig-zag retraction method to provide a homotopically equivalent but much smaller chain complex. The zig-zag retracted pure complex in Example 2.25 could, for example, be assigned a discrete vector field as illustrated in Figure 2.20:


(a) A discrete vector field on the contracted complex in Example 2.25

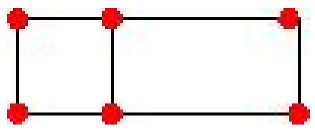

(b)

Figure 2.20: Zig-zag retract of Example 2.25

The following GAP code extract provides another example of how the retractions for pure complexes and discrete Morse theory method for calculating the critical cells of a regular CW space can be effectively combined. As part of this computation the *Cěch complex*, as explained in Definition 1.30 of the pure cubical complex is calculated. The *Cěch complex* of a pure cubical complex is a simplicial complex such that it contains one vertex for every cell in the pure complex, one edge for the non-empty intersection of any two cells in the pure complex, and in general there is one k-cell in the Cěch complex for the non-empty intersection of any $k + 1$ cells in the pure complex.

```
gap> A:=List([1..15],i->1);;

gap> A:=List([1..15],i->StructuralCopy(A));;

gap> A:=List([1..15],i->StructuralCopy(A));;

gap> A[6][6][6]:=0;;

gap> M:=PureCubicalComplex(A);;

gap> S:=CěchComplexOfPureCubicalComplex(M);;

gap> List([0..7],i->S!.nrSimplices(i));

gap> [3374,38040,122956,184168,153384,76664,21896,2736]

gap> Y:=SimplicialComplexToRegularCWSpace(S);;

gap> Size(Y);

603218

CriticalCellsOfRegularCWSpace(Y);time;

[[2, 19330], [0, 2140]]

29234
```

In the preceding segment the critical cells were calculated directly from the original complex, giving one 2-dimensional cell, the 19330th, and one 0-dimensional cell the 2140th, in just under half a minute. A contraction of the complex before assigning the discrete vector field is considerably faster.

```
gap> Size(M);

3374

gap> N:=ContractedComplex(M);;time;

28

gap> Size(N);

6
```

Once all cells with positive neighbourhoods have been removed, a discrete vector field can be assigned to the remaining cells in the space.

```
gap> S:=CĕchComplexOfPureCubicalComplex(N);;

gap> Y:=SimplicialComplexToRegularCWSpace(S);;

gap> Size(Y);

26

gap> C:=CriticalCellsOfRegularCWSpace(Y);; time;

0

C;

[[2,1],[0,5]]
```

Again we have one 2-dimensional cell and one 0-dimensional cell.

2.7 Singularities

Homological properties alone are not always sufficient to distinguish two spaces. In Figure 2.21 we have two spaces which are both homotopy equivalent to a point.

Let X and $Y \subset X$ be tessellated subspaces. The *complement* of Y is the tessellated subspace $X - Y$ which consists of the union of facets in X but not in Y. By the *boundary* of Y we mean the tessellated subspace δY consisting of the union of the facets f in Y such that $N_X(f) \cap X - Y \neq 0$. The tessellated subspace $Y - \delta Y$, we call the *interior* of Y. Size(Y) denotes the number of facets in Y.

Definition 2.29. Given a facet f in X and an integer $r \geq 1$ we define the *ball* of radius r centred at f to be the tessellated subspace $B_x(r, f) = N_X(B_X(r-1, f))$ where $B_X(1, f) = N_X(f)$, and the *sphere* of radius r centred at f to be the tessellated subspace $S_X(r, f) = \delta B_X(r, f)$.

Definition 2.30. Given an integer $r \geq 1$ and number $0 < \tau < 1$ we say that a facet f of the tessellated space $Y \subset X$ is (r, τ)-smooth if either f lies in the interior of Y or else the complement $S_X(r, f) - \delta Y$ consists of exactly two contractible path-components C_1, C_2

satisfying

$$\frac{\text{Size}(C_1) - \text{Size}(C_2)}{\text{Size}(C_1) + \text{Size}(C_2)} < \tau.$$

We say that $f \in Y$ is (r, τ)-*singular* if it is not (r, τ)-smooth. Note that to verify that there are two contractible path components it suffices to check that $\beta_i(S_X(r, f) - \delta Y) = 2$ if $i = 0$ and is 0 otherwise.

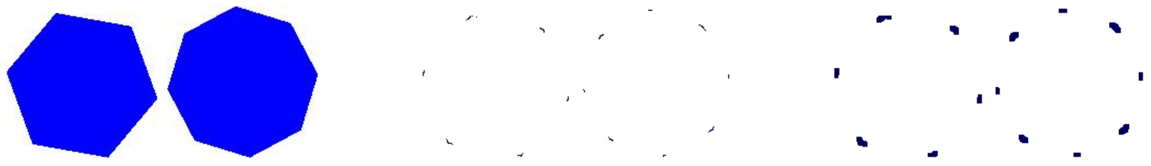


Figure 2.21: Two polygonal disks, their singularities, and thickened singularities

The two polygonal disks in Figure 2.21 were represented as 2-dimensional pure cubical complexes with 14316 and 13795 facets respectively. Shown are the pure cubical subcomplexes consisting of the $(5, 0.15)$-singular facets and thickenings of both of those subcomplexes. The persistence bar codes were constructed for a series of thickenings and showed that there are six and eight path components respectively for the two polygons.

3 Cubical Complexes

This chapter deals with subspaces of the tessellated space $\mathbb{R}^n_C$ whose structure is inherited from the lattice generated by n orthonormal basis vectors. A facet of $\mathbb{R}^n_C$ is combinatorially equivalent to an n-cube. In this chapter some theory is developed for abstract cubical complexes, cubical complexes and pure cubical complexes which are based on such tessellations.

Pure cubical complexes deal only with cubes of common dimension, which simplifies the representation and relevant algorithms. Cubical complexes deal with cubes of many dimensions, which makes the representation and algorithms more complicated than for pure cubical complexes, but are necessary because the distinction between edges, vertices, etc. needs to be made for homology computations etc. Abstract cubical complexes can prove more efficient in sparse or higher-dimensional data sets.

3.1 Pure cubical complexes

Recall that the n-dimensional cubical lattice L^n_C is an additive subgroup generated by n orthonormal vectors in $\mathbb{R}^n$, where any $v \in L^n_C$ determines a Dirichlet-Voronoi cell

$$D(v) = \{x \in \mathbb{R}^n : ||v - x|| \leq ||w - x|| \text{ for any } w \in L^n_C\}$$

from which Euclidean space $\mathbb{R}^n$ inherits the structure of a tessellated space whose facets are the cells $D(v)$. This tessellated space is denoted by $\mathbb{R}^n_C$.

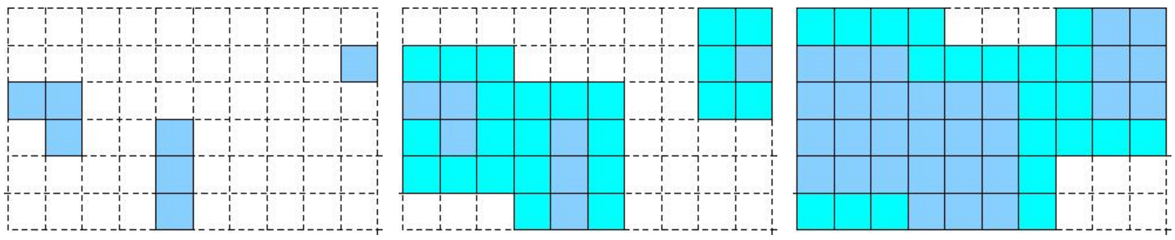
Definition 3.1. A *pure cubical space* X is a tessellated subspace of $\mathbb{R}^n_C$. The *dimensions* of X is a list of lengths, $[d_1, d_2, \ldots, d_n]$, where d_i is the maximal difference between the i^{th} coordinates of any two points in X. By applying a translation if necessary we can assume that

X is a union of translations of facets $D(v)$ where any v has only positive integer coordinates with respect to the given basis.

Definition 3.2. Recall our Definition 2.13 of a pure B-complex. An n-dimensional *pure cubical complex* K is a pure B-complex with orthonormal basis vectors, $(t_1, t_2, \ldots, t_n)$.

Let Γ be the set of all $\kappa = (\kappa_1, \kappa_2, \ldots, \kappa_n)$ where $\kappa_i \in \{-1, 0, 1\}$. The *neighbourhood* $N_K(k_\lambda)$ of an entry k_λ in an n-dimensional pure cubical complex K consists of all entries $k_{\lambda+\kappa}$ for $\kappa \in \Gamma$. The *neighbourhood* $N_X(x_\lambda)$ of a facet x_λ in an n-dimensional pure cubical space consists of the union of all facets $x_{\lambda+\kappa}$ in $\mathbb{R}^n_C$. Previously described algorithms for tessellated spaces can be applied to pure cubical complexes. In Example 3.3 we use Algorithm 2.17 to thicken a pure cubical complex K_0 and its corresponding pure cubical space X_0.

Example 3.3. An illustration of two iterations of Algorithm 2.17 over a pure cubical complex K_0.



(a) X_0 (b) X_1 (c) X_2

Figure 3.1: An illustration of `ThickenedPureCubicalComplex`

$$\begin{pmatrix} 0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&1\\ 1&1&0&0&0&0&0&0&0&0\\ 0&1&0&0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0 \end{pmatrix} \quad \begin{pmatrix} 0&0&0&0&0&0&0&0&1&1\\ 1&1&1&0&0&0&0&0&1&1\\ 1&1&1&1&1&1&0&0&1&1\\ 1&1&1&1&1&1&0&0&0&0\\ 0&1&1&1&1&1&0&0&0&0\\ 0&0&0&1&1&1&0&0&0&0 \end{pmatrix} \quad \begin{pmatrix} 1&1&1&1&1&0&0&0&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&0&0&0\\ 1&1&1&1&1&1&1&0&0&0 \end{pmatrix}$$

K_0 K_1 K_2

3.2 Cubical complexes

To calculate the homology of a pure cubical complex or space it is necessary, as with simplicial complexes, to determine which $(m-1)$-cells form the boundary of which m-cells.

As d-dimensional pure cubical complexes and pure cubical spaces restrict themselves to d-dimensional facets only, the notion of cubical complexes and cubical spaces is introduced. We use Example 3.4 as an illustration of the process of converting a pure cubical complex, which represents a collection of facets, into the corresponding cubical complex where the entries represent 0-cubes, 1-cubes and 2-cubes.

Example 3.4. In this example a pure cubical complex K, where $X = |K|$, is converted to a cubical complex K' using Algorithm 3.9, detailed below.

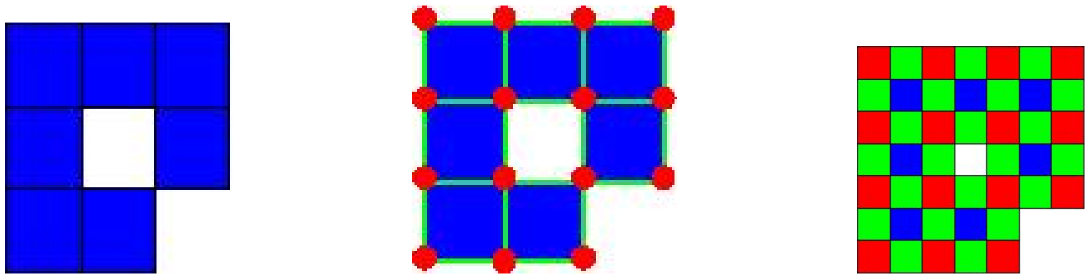

(a) X (b) X' (c) X''

Figure 3.2: A pure cubical space X, the separation X' of pure cubes into 0-cubes, 1-cubes and 2-cubes, and as a cubical space X'' which illustrates how each cell corresponds to an entry '1' in the cubical complex K'.

$$\underset{K}{\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}} \qquad \underset{K'}{\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}}$$

The pure cubical complex consists of 7 facets, whereas the corresponding cubical complex consists of 44 cells: seven 2-cells (blue), twenty-two 1-cells (green) and fifteen 0-cells (red). Implementations for pure cubical complexes are usually more efficient, though there are some cases in which cubical complexes must be used. Calculating the chain complex, for example, requires the use of cubical complexes, but in general it is preferable to use pure cubical complexes where possible. Not all cubical complexes have corresponding pure cubical complexes.

Definition 3.5. A *cubical complex* K of dimension d is a binary array with index λ ranging over the set

$$\Lambda = \{1, 2, \ldots, 2n_1 + 1\} \times \{1, 2, \ldots, 2n_2 + 1\} \times \ldots \times \{1, 2, \ldots, 2n_d + 1\} \subset \mathbb{N}^d$$

that arises from a CW-subspace of a pure cubical complex as illustrated in Example 3.4.

The *dimensions* of K are the integer vector $(2n_1 + 1, 2n_2 + 1, \ldots, 2n_d + 1)$. There is one axiom: if some entry $k_\lambda = 1$ then the entry $k_{\lambda'} = 1$, where $\lambda' \in \Lambda$ is obtained by adding ± 1 to an even entry in the index λ.

We define $\xi(\lambda)$ to be the number of even entries in λ.

Definition 3.6. A *cubical space* $X \subset \mathbb{R}^n_C$ is a CW-subspace. The cell x_λ of X is a $\xi(\lambda)$-cube of the d-cubes $x_{\lambda+\kappa}$ present in X in $\mathbb{R}^n_C$ where $\kappa \in [-1, 0, 1]^n$ such that $\xi(\kappa + \lambda) = d$. By $[-1, 0, 1]^n$ we mean all possible lists of length n with entries chosen from $[-1, 0, 1]$.

Example 3.7. A cell $x_{(3,5,7)}$ in $\mathbb{R}^3_C$ is a vertex of X. It is a 0-cell of the 1-cells $x_{(2,5,7)}, x_{(3,4,7)}, x_{(3,5,6)}, x_{(3,5,8)}, x_{(3,6,7)}, x_{(4,5,7)}$, of the 2-cells $x_{(2,4,7)}, x_{(2,5,6)}, x_{(2,5,8)}, x_{(2,6,7)}, x_{(3,4,6)}, x_{(3,4,8)}, x_{(3,6,6)}, x_{(3,6,8)}, x_{(4,4,7)}, x_{(4,5,6)}, x_{(4,5,8)}, x_{(4,6,7)}$, and of the 3-cells $x_{(2,4,6)}, x_{(2,4,8)}, x_{(2,6,6)}, x_{(2,6,8)}, x_{(4,4,6)}, x_{(4,4,8)}, x_{(4,6,6)}, x_{(4,6,8)}$.

Figure 3.3 shows twelve 2-cells which share a common vertex.

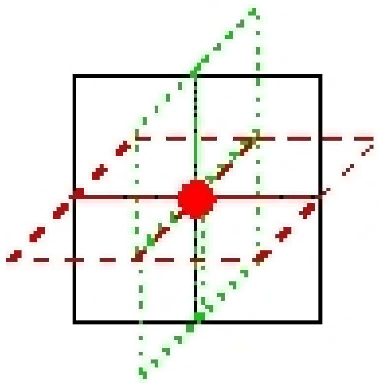


Figure 3.3: A 0-cell of twelve 2-cells in R^3.

An n-dimensional cubical complex K can be geometrically realised as a cubical space $|K| = X \subset \mathbb{R}^n_C$ where $x_\lambda = 1$ if and only if $k_\lambda = 1$ and the $\xi(\lambda)$-cell

$$D(v) = \{x \in \mathbb{R}^n \text{ where } ||v - x|| \leq ||v \pm w - x|| \text{ for any } w \in \{0,1\}^n\} \text{ with } \xi(v) = \xi(\lambda),$$

where $v = (\lambda_1 t_1 + \lambda_2 t_2 + \ldots + \lambda_n t_n)$ and the t_i are the orthonormal basis vectors.

Example 3.8. The cubical complex with binary array K

$$K = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

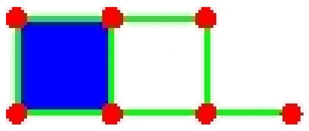


Figure 3.4: X: the corresponding cubical space for K.

has one 2-cell at $k_{\{2,2\}}$, seven 1-cells at $k_{\{1,2\}}, k_{\{1,4\}}, k_{\{1,6\}}, k_{\{2,1\}}, k_{\{2,3\}}, k_{\{2,5\}}$, $k_{\{3,2\}}, k_{\{3,4\}}, k_{\{3,6\}}$ and seven 0-cells at $k_{\{1,1\}}, k_{\{1,3\}}, k_{\{1,5\}}, k_{\{3,1\}}, k_{\{3,3\}}, k_{\{3,5\}}, k_{\{3,7\}}$.

While they have a similar representation, it is important to note the difference between a pure cubical complex or pure cubical space and a cubical complex or cubical space respectively. In a d-dimensional pure cubical complex, k_λ represents a d-cell, while k_λ in a cubical complex represents a $\xi(\lambda)$-cell. Any d-dimensional pure cubical complex K can be converted to a cubical complex K' as illustrated in Algorithm 3.9.

Algorithm 3.9. Pure Cubical Complex to Cubical Complex

Input: A pure cubical complex K.

Output: A cubical complex K'.

Procedure:

```
    if $K$ has dimensions $(n_1, n_2, \dots, n_d)$ then let $K'$ be a cubical complex with
    dimensions $(2n_1 + 1, 2n_2 + 1, \dots, 2n_d + 1)$.
    for $\lambda$ in $\Lambda$ do
      for $\kappa \in [-1, 0, 1]^n$ do
        Set $k'_\theta = 1$ precisely if $k_\lambda = 1$ and $\theta = 2\lambda + \kappa$.
      end for
    end for
```

An m-cell k_λ in an n-dimensional cubical complex with coordinates $(\lambda_1, \lambda_2, \dots, \lambda_n)$ has m entries λ_i which are even integers. The boundary of such an m-cell is the sum of the $(m-1)$-cells of K which are obtained by adding 1 to exactly one even entry in k_λ less the sum of the $(m-1)$-cells of K obtained by subtracting 1 from exactly one even entry in k_λ.

To calculate the homology of the pure cubical complex K, it is first converted to the cubical complex K'. The free abelian groups in the sequence for the chain complex are given by an ordered list of the 2-cells, 1-cells and 0-cells. In Figure 3.2 one can see that the boundary of the dark blue 2-cells, such as those with coordinates $(2, 2), (2, 4), (3, 2)$ are the respective adjacent green 1-cells, such as $(1, 2), (2, 1), (3, 6)$. The boundary of the 1-cells are the red 0-cells, which have only odd coordinates, such as $(1, 1), (3, 7), (7, 5)$. The homology is calculated from the chain complex as described in chapter 1.

3.3 Abstract cubical complexes

Let S be a set. An S-pair is a pair $P = (P_1, P_2)$ of subsets of S such that $P_2 \subseteq P_1$. Associated to an S-pair P is the lattice of all subsets of P_1 containing P_2; inclusion is the partial order. Given an S-pair P and an element $x \in P_1 \backslash P_2$, define

- $\delta_x^+(P) = (P_1, P_2 \cup \{x\})$
- $\delta_x^-(P) = (P_1 \backslash \{x\}, P_2)$

Definition 3.10. An *abstract cubical complex* [17] consists of an ordered set S together with a set J of S-pairs satisfying the following condition:

$$P \in J \Rightarrow \delta_x^+(P) \in J \text{ and } \delta_x^-(P) \in J \text{ for all } x \in P_1 \backslash P_2.$$

A pair P in J has dimension $|P_1| - |P_2|$, where $|P_1|$ is the cardinality of $|P_1|$.

Definition 3.11. The *chain complex of an abstract cubical complex* is the sequence of free abelian groups

$$\ldots \to C_n(J) \xrightarrow{\delta_n} C_{n-1}(J) \xrightarrow{\delta_{n-1}} \ldots \xrightarrow{\delta_1} C_0(J)$$

where $C_n(J)$ is the free abelian group on the n-dimensional pairs in J and the boundary homomorphism $\delta_n : C_n(J) \to C_{n-1}(J)$ is defined on each n-dimensional pair $P = (P_1, P_2)$ by the formula

$$\delta_n(P) = \sum_{x_i \in P_1 \backslash P_2} (-1)^i (\delta_{x_i}^+(P) - \delta_{x_i}^-(P)).$$

Lemma 3.12. *For any pair of entries x, y in an ordered set S, the resulting set after the removal of x and y is the same, irrespective of the order in which they were removed. Therefore, ignoring signs,*

$$\delta_y^+ \delta_x^+ = \delta_x^+ \delta_y^+,$$

$$\delta_y^- \delta_x^+ = \delta_x^+ \delta_y^-,$$

$$\delta_y^+ \delta_x^- = \delta_x^+ \delta_y^-,$$

$$\delta_y^- \delta_x^- = \delta_x^- \delta_y^- .$$

Theorem 3.13. *The composition of any two consecutive boundary homomorphisms in the chain complex of an abstract cubical complex $\delta_n \circ \delta_{n+1} = 0$.*

Proof. Working mod 2 and ignoring signs, we can write the boundary homomorphism as

$$\delta_x(P) = \sum_{x \in P_1/P_2} (\delta_x^+(P) - \delta_x^-(P)).$$

$$\delta_y(\delta_x(P)) = \sum_{x \in P_1/P_2, y \neq x} \delta_y^+ \delta_x^+(P) - \delta_y^+ \delta_x^-(P) - \delta_y^- \delta_x^-(P) + \delta_y^- \delta_x^+(P).$$

Since the presence of the pair of entries x, y implies the presence of y, x, we may infer from our lemma 3.12 that each of $\delta_y^+ \delta_x^+(P)$, $\delta_y^+ \delta_x^-(P)$, $\delta_y^- \delta_x^-(P)$, $\delta_y^- \delta_x^+(P)$ occurs exactly twice and signs can be chosen so as to ensure that the composition is 0. □

The homology, Betti numbers and Euler Characteristic of an abstract cubical complex are those of the chain complex of the abstract cubical complex, and are calculated in the same manner as for simplicial complexes.

Example 3.14. Calculating the homology of an abstract cubical complex.
Consider the 2-simplex with vertices a, b, c. Let S be the set of all subsets of these vertices. Let K be the abstract cubical complex consisting of the ordered set S together with the set of S-pairs of dimension 2 or less. The six pairs of dimension 2, for example

are:$(\{a,b,c\},\{b\}),(\{a,b,c\},\{a\}),(\{a,b,c\},\{c\}),(\{a,b\},\emptyset),(\{a,c\},\emptyset)$ and $(\{b,c\},\emptyset)$.

$$\begin{tikzcd}
 & \{a,b,c\} & & \{a,b\} \arrow[ll] \\
\{b,c\} \arrow[ur] & & \{b\} \arrow[ll] \arrow[ur] & \\
 & \{a,c\} \arrow[uu] & & \{c\} \arrow[ll] \arrow[uu] \\
\{a\} \arrow[uu] \arrow[ur] & & \emptyset \arrow[ll] \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

K is represented by the chain complex

$$0 \to \mathbb{Z}^6 \xrightarrow{\delta_2} \mathbb{Z}^{12} \xrightarrow{\delta_1} \mathbb{Z}^8$$

where

$$\delta_2 = \begin{pmatrix}
-1 & 1 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & -1 & 1
\end{pmatrix},$$

$$\delta_1 = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}.$$

The $(i,j)^{th}$ entry in δ_n represents the orientation of the i^{th} $(n-1)$-dimensional cell in j^{th} n-dimensional cell.

We can then calculate that

$$H_n(K) = \begin{cases} \mathbb{Z} & \text{if} \quad n = 0, 2 \\ 0 & \text{if} \quad n \neq 0, 2 \end{cases}.$$

The 0^{th} and 2^{nd} Betti numbers are 1 and the others are 0. The Euler Characteristic is 2.

Any cubical complex can be faithfully represented as an abstract cubical complex. We present the algorithm used as a proof. First we define a number of functions used within the algorithm.

Remark: any abstract cubical complex can conversely be expressed as a cubical complex, if the cubical complex is of sufficiently high dimension.

Definition 3.15. A *binary Gray code reflection* [21] $\mathbb{G}_n$ is a bijection from $\{0,1\}^n$ given by

$$\mathbb{G}_n(x) = \begin{cases} 0 & \text{if } \ n = 1 \text{ and } x = 0, \\ 1 & \text{if } \ n = 1 \text{ and } x = 1, \\ 0\mathbb{G}_{n-1}(x) & \text{if } \ 0 \leq x \leq 2^{n-1} - 1, \\ 1\mathbb{G}_{n-1}(2^n - 1 - x) & \text{if } \ 2^{n-1} \leq x \leq 2^n - 1 \end{cases}$$

This then gives that for any two consecutive integers $i, j < 2^n$ the strings $\mathbb{G}_n(i)$ and $\mathbb{G}_n(j)$ are of length n and differ by 1 in exactly one component and are equal elsewhere.

Define $\rho : \{0,1\}^n \to \mathbb{P}(\{1,2,\ldots,n\})$ where $\mathbb{P}$ is the power set. If $\underline{s} = (s_1, \ldots, s_n)$ then $\rho(\underline{s})$ is defined by $i \in \rho(\underline{s})$ if and only if $s_i = 1$, for $1 \leq i \leq n$. This function maps a binary string to a set of integers, with each integer indicating the position of a "1" in the binary string.

Define γ to be a function which concatenates multiple strings of integers into a single string.

Theorem 3.16. *Every cubical complex can be faithfully represented as an abstract cubical complex (ie. they have isomorphic chain complexes).*

Proof. Let $K \subseteq \mathbb{R}^n$ be a cubical complex with 1-skeleton K^1. An embedding using binary Gray code reflection can be used to embed K^1 into the graph of the cube $\mathbb{I}^N$ for some N. Here $\mathbb{I}^n$ can be thought of as the poset of subsets of $\{1,2,\ldots,N\}$. The vertices of any face F of K correspond to an interval in the poset $\mathbb{I}^n$. This interval has upper bound $max(F) \subseteq \{1,2,\ldots,N\}$, say, and lower bound $min(F) \subseteq \{1,2,\ldots,N\}$.

Thus we can represent K as the abstract cubical complex with

Vertex set $= \left\{ \text{ those vertices of } \mathbb{I}^n \text{ lying in the embedded image of } K^1 \right\}$

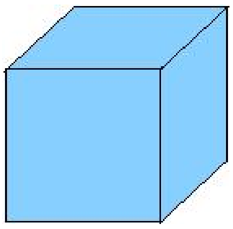
Pairs $= \left\{ \{max(F), min(F)\} : F\text{is a cube in}K. \right\}$ $\square$

Example 3.17. The following extract of GAP code illustrates some simple homology calculations for a 3-cube with some alterations, with creation of the abstract cubical complex both from a selection of pairs of subsets and by conversion from a cubical complex.

```
gap> A:=[ [ [ 1, 2, 3, 4 ], [ 1, 2, 3, 4 ] ],
[ [ 1 ], [ 1 ] ], [ [ 1, 2, 3, 4 ], [ 1 ] ] ];;
gap> B:=MakePairsToAbstractCubicalComplex(A);;
```

(This is further described in Chapter 5, but in this case it suffices to note that it creates an abstract cubical complex, which corresponds to a solid 3-dimensional cube.)

```
gap> C:=ChainComplexOfAbstractCubicalComplex(B);;
gap> Homology(C);
[[ 0 ],[ ],[ ],[ ]]
```



B is the union of the 3-cube

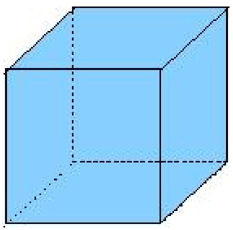
$[[1, 2, 3, 4], [1]]$

and all cells in its boundary.

```
gap> Remove(B!.pairs[4],1);;
```

(This removes the first (and only) three-dimensional cell from the complex, leaving a hollow 3-dimensional cube.)

```
gap> C:=ChainComplexOfAbstractCubicalComplex(B);;
gap> Homology(C);
[[ 0 ],[ ],[ 0 ],[ ]]
```

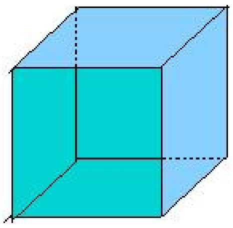


$B - ([[1, 2, 3, 4], [1]])$

```
gap> Remove(B!.pairs[3],3);;
```

(This removes a 2-dimensional cell.)

```
gap> C:=ChainComplexOfAbstractCubicalComplex(B);;
gap> Homology(C);
[[ 0 ],[ ],[ ],[ ]]
```



$B - ([[1, 2, 3, 4], [1]] + [[1, 2, 4], [1]])$

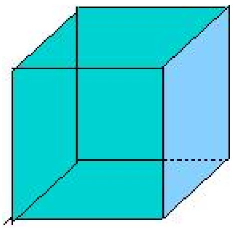
```
gap> Remove(B!.pairs[3],1);;
```

(This removes another 2-dimensional cell.)

```
gap> C:=ChainComplexOfAbstractCubicalComplex(B);;

gap> Homology(C);

[[ 0 ],[ 0 ],[ ],[ ]]
```



$B - ([[1,2,3,4],[1]] + [[1,2,4],[1]] + [[1,2,3,4],[1,2]])$

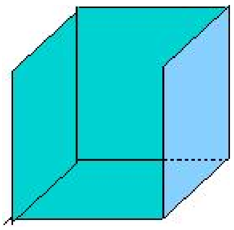
```
gap> Remove(B!.pairs[2],8);;
```

(This removes an edge.)

```
gap> C:=ChainComplexOfAbstractCubicalComplex(B);;

gap> Homology(C);

[[ 0 ],[ ],[ ],[ ]]
```



$B - ([[1,2,3,4],[1]] + [[1,2,4],[1]] + [[1,2,3,4],[1,2]] + [[1,2,4],[1,2]])$.

The following GAP code extract illustrates the creation of an abstract cubical complex from a cubical complex.

```
gap> D:=PureCubicalComplex([[[1]]]);;

gap> F:=PureCubicalComplexToCubicalComplex(D);;

gap> G:=CubicalComplexToAbstractCubicalComplex(F);;

gap> H:=ChainComplexOfAbstractCubicalComplex(G);;

gap> Homology(H);

[[ 0 ],[ ],[ ],[ ]]

gap> Size(G!.pairs[1]);

8

gap> Size(G!.pairs[2]);

12

gap> Size(G!.pairs[3]);

6

gap> Size(G!.pairs[4]);

1

gap> F!.binaryArray[2][2][2]:=0;;

gap> G:=CubicalComplexToAbstractCubicalComplex(F);;

gap> H:=ChainComplexOfAbstractCubicalComplex(G);;

gap> Homology(H);

[[ 0 ],[ ],[ 0 ],[ ]]
```

4 Permutahedral Complexes

In this section we introduce and develop some theory for tessellated complexes based on permutahedra. A selection of reasons are listed here briefly to illustrate our motivation for using complexes with a permutahedral tessellation. We elaborate on some of these reasons later in the chapter.

1. The contracting homotopy approach discussed in Chapter 2 for pure B-complexes is practical up to dimension 3 in the category of pure cubical complexes, and up to dimension 4 in the category of pure permutahedral complexes.

   Remark: Working with non-pure complexes, homotopy theoretic techniques can be used in higher dimensions. However, such techniques are often slower than those for pure complexes.

2. It is relatively simple to convert from readily available cubical to permutahedral image formats, especially if we are only interested in analysing topological properties as there is less need to account for skewing and stretching. Though the homology of a pure cubical complex and pure permutahedral complex with the same binary array need not be isomorphic, experiment has shown that the persistent homology through thickenings of each is similar.

3. There in no ‘connectivity ambiguity’ with permutahedral data, unlike with cubical data. This is explained in detail on page 54.

4. A permutahedron has fewer neighbours than a cube, yielding computational advantage for algorithms which require the manipulation of these neighbours.

5. Some data, such as medical data [31][32][4], images from Fuji's F30 super CCD camera, and data for some computer game graphics, are already produced hexagonally [41][42].

4.1 The permutahedron

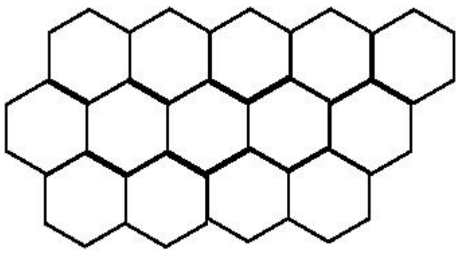


Figure 4.1: Hexagons can be used to tessellate a 2-dimensional plane.

The permutahedron $(n-1)$-dimensional permutahedron is an $(n-1)$-dimensional polytope embedded in n-dimensional space. Its vertices are formed by permuting the coordinates of the vector $(1, 2, \ldots, n)$. Two vertices u and v of a permutahedron are connected by an edge if $(u_1, u_2, \ldots, u_n)$ can be obtained by permuting exactly two of the coordinates v_i and v_j in $(v_1, v_2, \ldots, v_n)$ and $|v_i - v_j| = 1$.

Example 4.1. The permutahedron of dimension 2 is a hexagon embedded in $\mathbb{R}^3$, whose vertices have coordinates $(1, 2, 3), (1, 3, 2), (2, 3, 1), (3, 2, 1), (3, 1, 2), (2, 1, 3)$. See Figure 4.2.

The 3-dimensional permutahedron is a truncated octahedron. [34] Its vertices are the permutations of the integers $1 \ldots 4$.

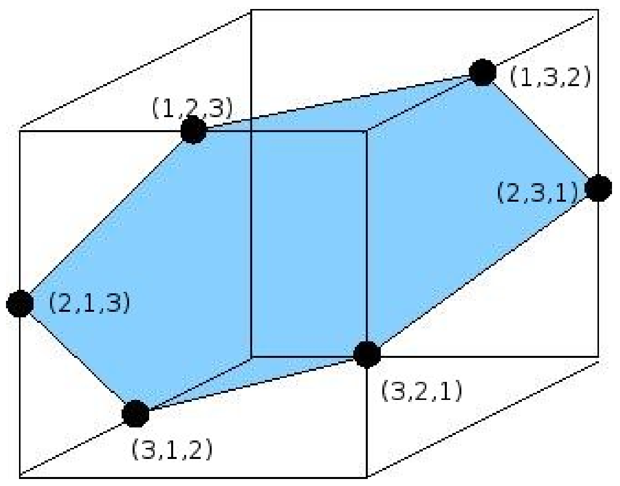


Figure 4.2: A 2-dimensional permutahedron is a hexagon embedded in $\mathbb{R}^3$.

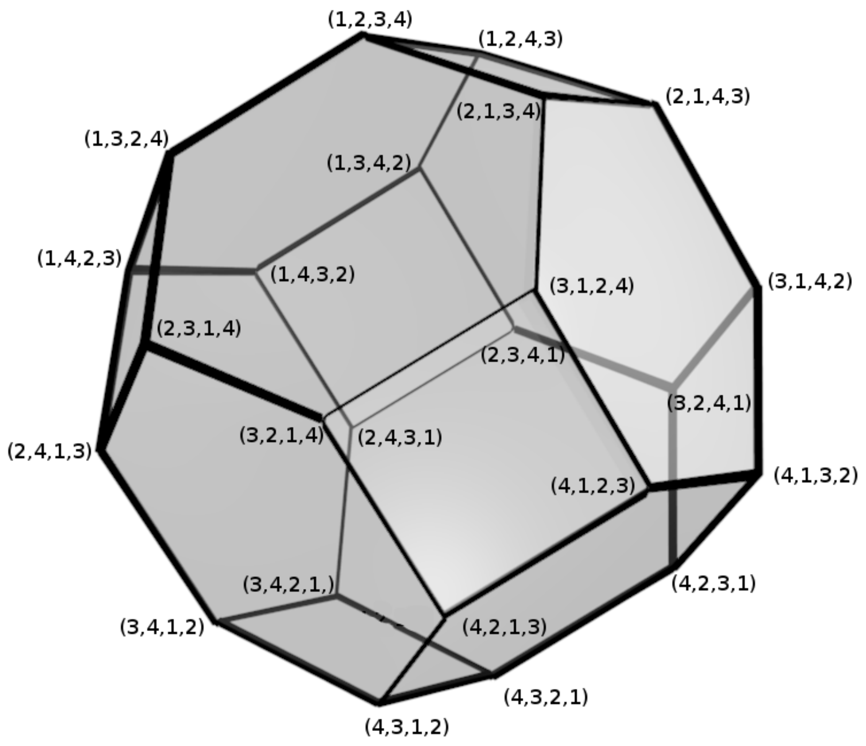


Figure 4.3: A 3-dimensional permutahedron is a truncated octahedron embedded in 4-dimensional space. Its vertices are the permutations of the integers 1..4. [22]

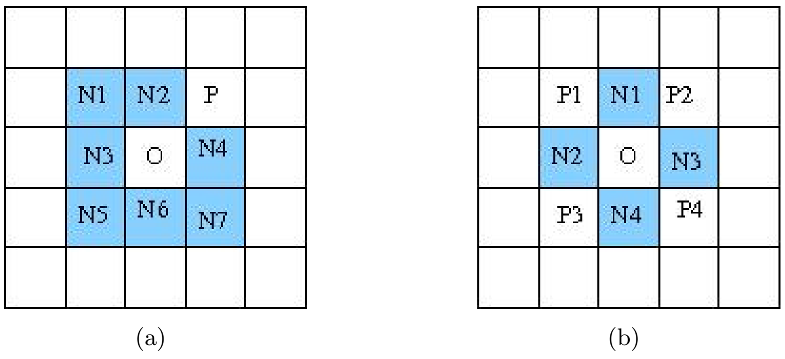
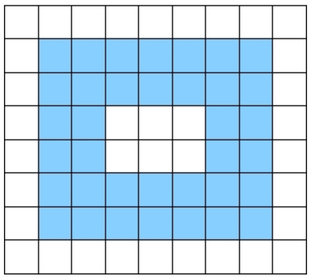
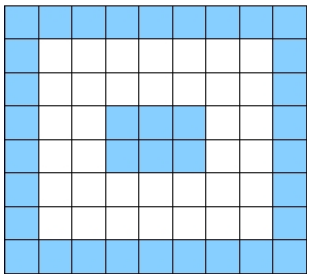


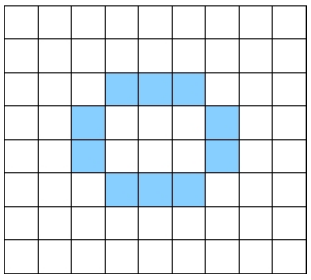
Figure 4.4: Connectivity paradox

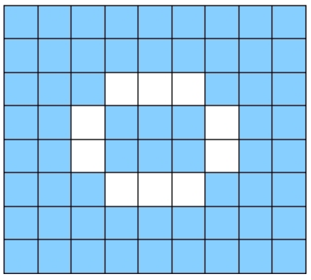
There is some debate, from an engineering point of view, in the pure cubical context, over whether two cells which share only a vertex are actually connected. See Figure 4.4*(a)*. If it is taken that a cell has eight neighbours other than the cell itself, then the cell O in 4.4*(a)* has a closed boundary. But P is a neighbour of O also, meaning that O is connected to the region outside the shape. Similarly, if it is taken that a cell has four neighbours other than the cell itself, then as illustrated in Figure 4.4*(b)*, the cell O has a boundary of four cells, $N1, N2, N3, N4$ which are unconnected to one another, which would imply O were connected to the region outside the 'annulus', yet none of the outside cells P_i are connected to O. Throughout this thesis we have adopted the usual convention [43] that cells which are present in some space X have eight neighbours excluding the cell itself, whereas any cell in R^n but not in X has four neighbours other than itself. It is worth noting that there is no such ambiguity in the permutahedral case. If two n-dimensional permutahedra are connected they share an $n-1$-dimensional cell. This gives that for any space X, in the permutahedral case, we know that $Comp(Cont(X))$ is homotopy equivalent to $Cont(Comp(X))$, where $Comp(X)$ is the complement of X and $Cont(X)$ is a homotopy retract of X. This is not true for cubical complexes. See Figs. 4.5 and 4.6.

If we wish to calculate the fundamental group of a link diagram, for example, in the pure cubical case we must find the complement of the original diagram and then contract that before calculating the critical cells of a homotopy equivalent CW-space. In the pure permutahedral case, however, we can first contract the original diagram, leaving us with a smaller CW-space whose complement we can then use. See Chapter 6.5 for an example.


(a) X


(b) $Comp(X)$


(c) $Cont(X)$


(d) $Comp(Cont(X))$

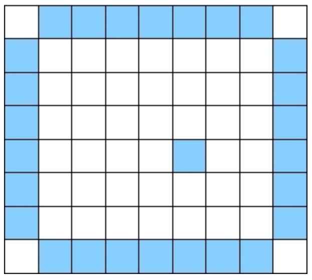
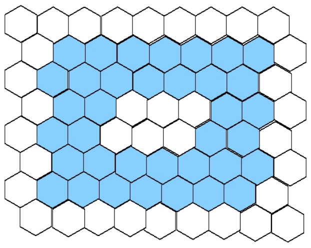
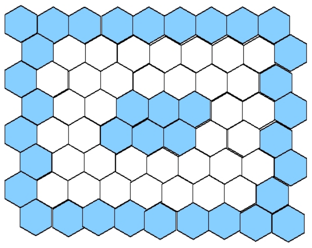
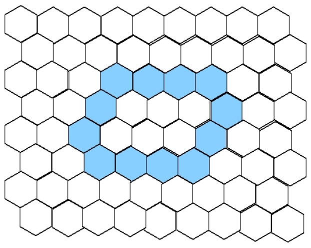
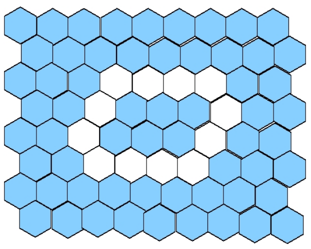

(e) $Cont(Comp(X))$

Figure 4.5: $Cont(Comp(X)) \not\simeq Comp(Cont(X))$


(a) X


(b) $Comp(X)$


(c) $Cont(X)$


(d) $Comp(Cont(X))$

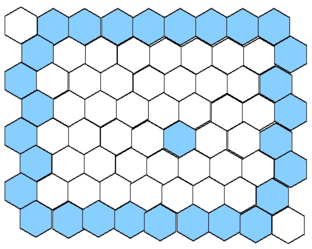

(e) $Cont(Comp(X))$

Figure 4.6: $Cont(Comp(X)) \simeq Comp(Cont(X))$

An n-dimensional permutahedron has $2^{n+1}-1$ neighbours, including itself, and permutahedral tessellations can thus be expected to have certain advantages over cubical tessellations for functions which use the neighbourhood of cells. The function `ThickenedPureCubical-Complex` in 3-dimensions requires that for every f in a pure cubical complex X every entry in $N_{X'}(f) = 1$, where X' is the thickened complex. So for each individual facet, 27 neighbours must be ascertained to have the value '1'. In the pure permutahedral case the corresponding number is 15. The difference becomes more pronounced as the dimension increases as seen in Table 4.1. In general an n-dimensional cube will have 3^n neighbours, whereas an n-dimensional permutahedron will have $2^{n+1} - 1$.

Table 4.1: Number of neighbours of a n-cell in a pure complex of dimension n.

n	Cubical	Permutahedral
1	3	3
2	9	7
3	27	15
4	81	31
5	243	63

Example 4.2. The following extract of GAP code illustrates a greater efficiency of the `ThickenedPurePermutahedralComplex` function compared to `ThickenedPureCubicalCom-plex`. Here a 3-dimensional image constructed by layering fifty copies of the binary array of K, where $X = |K|$, see Figure 4.7, on top of one another is thickened both cubically and permutahedrally.

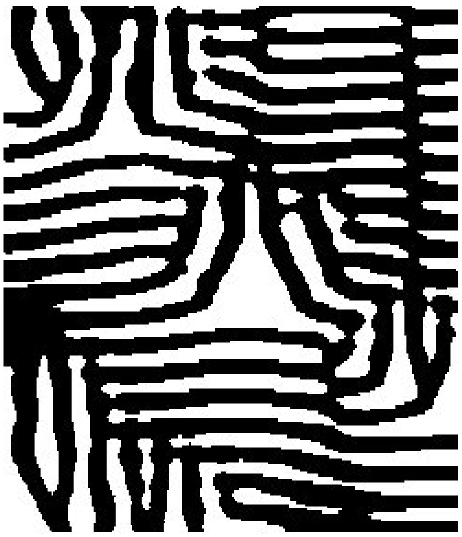


Figure 4.7: $|X|$
[2]

```
gap> L:=List([1..50],i->StructuralCopy(K));;

gap> P:=PurePermutahedralComplex(L);;

gap> C:=PureCubicalComplex(L);;

gap> ThickenedPurePermutahedralComplex(P);time;

Pure permutahedral complex of dimension 3.

21069

gap> ThickenedPureCubicalComplex(C);time;

Pure cubical complex of dimension 3.

86061
```

Furthermore, the smaller number of neighbours, and thus the smaller number of possible neighbourhoods for the pure permutahedral case makes it possible to store a list of all positive neighbourhoods (recall that a cell with positive neighbourhood is one which can be removed without affecting its neighbourhood's homology) in four dimensions, whereas in the cubical case this was only possible in up to three dimensions. Table 4.2 indicates the number of possible neighbourhoods and positive neighbourhoods for both the pure cubical and pure permutahedral complexes. This optimal redundancy test enables us to efficiently compute homotopy retracts in four dimensions in the permuathedral case.

Table 4.2: Possible and positive neighbourhoods of a n-cell in a pure complex of dimension n.

n	Possible neighbourhoods in the cubical case	Positive neighbourhoods in the cubical case	Possible neighbourhoods in the permutahedral case	Positive neighbourhoods in the permutahedral case
1	4	2	4	2
2	256	116	64	30
3	67108864	41123720	16384	7500
4	$1.20892582 \times 10^{24}$	-	1073741824	280694770
5	$7.06738826 \times 10^{72}$	-	$4.61168602 \times 10^{18}$	-

4.2 $\mathbb{R}^n_P$

If we identify $\mathbb{R}^n$ with the hyperplane in $\mathbb{R}^{n+1}$ consisting of all vectors whose components sum to zero, and let P be the abelian group generated by the columns of the $(n+1) \times (n+1)$ matrix

$$\begin{pmatrix} -n & 1 & 1 & \dots & 1 \\ 1 & -n & 1 & \dots & 1 \\ \vdots & & & & \\ 1 & 1 & 1 & \dots & -n \end{pmatrix}$$

then the facets of the tessellated space $\mathbb{R}^n_P$ are combinatorially equivalent to an n-permutahedron.

Example 4.3. The permutahedron of dimension 2:

The vectors $(-2,1,1)$ and $(1,-2,1)$ generate $\mathbb{R}^2_P$. A pure permutahedral complex K whose binary array,

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

has entries at $(1,2),(2,2),(3,1),(3,2)$, and $(3,4)$, will therefore have a corresponding space $X = |K|$ whose entries are Dirichlet-Voronoi cells centred at

$$\begin{gathered}
1 \times (-2,1,1) + 2 \times (1,-2,1) = (0,-3,3) \\
2 \times (-2,1,1) + 2 \times (1,-2,1) = (-2,-2,4) \\
3 \times (-2,1,1) + 1 \times (1,-2,1) = (-5,1,4) \\
3 \times (-2,1,1) + 2 \times (1,-2,1) = (-4,-1,5), \text{ and} \\
3 \times (-2,1,1) + 4 \times (1,-2,1) = (-2,-5,7).
\end{gathered}$$

4.3 Pure permutahedral complexes

Definition 4.4. Recall our definition of a pure B-complex from Chapter 2. An n-dimensional *pure permutahedral complex* is a pure B-complex whose basis vectors are defined by $\underline{T} = (t_1, t_2, \ldots, t_n)$ where each t_i is of length $(n+1)$ and has coordinates which sum to 0 and are equivalent *mod* $(n+1)$. The calculation of the neighbours in a pure permutahedral complex is explained in the next section.

Definition 4.5. The lattice $L_P \subset \mathbb{R}^n_P$ is the set of the centres of all facets in the permutahedral tessellation of R^n. A *pure permutahedral space* X is some tessellated subspace of $\mathbb{R}^n_P$.

Example 4.6. The 2-dimensional pure permutahedral complex with binary array

$$K = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

indicates that the corresponding pure permutahedral space $X = |K| \subset \mathbb{R}^3$ with basis $\underline{T} = \{(-2,1,1),(1,-2,1)\}$ would have facets with centres at $(0,-3,3),(1,-5,4),(-3,0,3),$ $(-1,-4,5),(-5,1,4),(-4,-1,5)$.

4.4 Neighbours in $\mathbb{R}_P^n$

An n-dimensional permutahedron has $2^{n+1}-1$ neighbours including itself. It shares an $(n-1)$-dimensional cell with each neighbour other than itself. The centre of this shared $(n-1)$-cell is half-way between the two neighbours, so we can find the coordinates of the centres of the neighbours using the centres of $(n-1)$-dimensional faces.

The vertices of the n-dimensional permutahedron are given by all permutations of $\{1,\dots,n+1\}$ and two of these vertices $u=(u_1,u_2,\dots,u_n), v=(v_1,v_2,\dots,v_n)$ are connected by an edge if v can be obtained by swapping u_i and u_j, where $|u_i-u_j|=1$.

The 2-dimensional case we could calculate as follows: A 2-dimensional permutahedron P shares a 1-dimensional cell with 6 neighbours. The vertices of P are $(1,2,3),(1,3,2),(2,3,1),(3,2,1),(3,1,2),(2,1,3)$, and are connected as illustrated in Figure 4.2.

The centres of the 1-cells are thus $(1,2.5,2.5),(1.5,3,1.5),(2.5,2.5,1),(3,1.5,1.5),$
$(2.5,1,2.5),(1.5,1.5,3)$, and so the corresponding neighbours of P which is centred at $(2,2,2)$ are $(0,3,3),(1,4,1),(3,3,0),(4,1,1),(3,0,3),(1,1,4)$.

Using the basis in Example 4.3; $t_1=(1,1,-2), t_2=(-1,2,-1)$, we can describe the centre of each neighbour as $(2,2,2)+n$ where $n\in N$ and $N=\{-t_1-t_2,-t_1,-t_2,t_1+t_2,t_1,t_2,0\}$.

For higher dimensions it is more difficult to visualise and, thus, capture which vertices are in an $n-1$-dimensional face. Choosing our basis $\underline{T}$ for $\mathbb{R}_P^d$ such that $t_1=(-d,1,\dots,1), t_2=(1,-d,1,\dots,1),\dots,t_d=(1,\dots,1,-d,1)$, where each t_i is of length $d+1$, gives a convenient description of the neighbours of a d-permutahedron with coordinates $(p_1,\dots,p_d)$; such that they can be described by $(p_1,\dots,p_d)\pm n\in N$ where $n\in\{\lambda_1t_1+\lambda_2t_2+\ldots+\lambda_dt_d\}$ where $\lambda_i\in\{0,1\}$.

Remark: We abuse notation and use "the permutahedron P" and "the permutahedron with centre P" interchangeably. In general, for any n-dimensional permutahedron P, its neighbourhood $N(P)$ is given by $P+n$ for $n\in N$.

Definition 4.7. Two facets p,q in X are said to be *incident* if $p\in N_X(q)$.

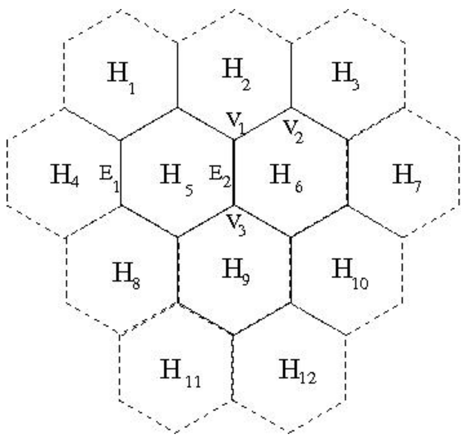


Figure 4.8: A 2-dimensional pure permutahedral space

4.5 Chain complex

An n-dimensional facet of a pure permutahedral complex does have $(n-1)$-dimensional cells in its boundary, but unlike in the cubical case, their structure is not consistent. A 3-dimensional permutahedral cell, for example, has both six-sided and four-sided 2-dimensional faces. For this reason we do not have the notions of a permutahedral complex or abstract permutahedral complex analogous to those for the cubical case. Instead, we express the k-cells in an n-dimensional pure permutahedral complex as an intersection of $n+1-k$ mutually incident permutahedral cells.

Definition 4.8. A k-dimensional cell in an n-dimensional pure permutahedral complex K is a subset containing $n+1-k$ pairwise incident facets in $\mathbb{R}^n_P$, at least one of which lies in $X = |K|$. The *boundary of a k-dimensional cell* which is the subset of facets $p_1, p_2, \ldots, p_{n+1-k}$ consists of the $(k-1)$-dimensional cells which are the intersections of $p_1, p_2, \ldots, p_{n+1-k}$ and a facet p_y which is incident with each of them.

Example 4.9. In Figure 4.8 the pure permutahedral space $X \subset R^2_P$ comprises the three 2-cells H_5, H_6, H_9. An edge e which lies in X is expressed as the intersection of two pairwise incident facets in R^2_P, both of which must contain e and at least one of which must be in X. The edge E_2, for example, is the intersection of the 2-cells H_5, H_6, both of which are in X. The edge E_1 is described as the intersection of H_4 and H_5 even though H_4 is not in X. Similarly the vertices V_1,V_2 and V_3 are described as the intersections of $\{H_2, H_5, H_6\}$, $\{H_2, H_3, H_6\}$ and

$\{H_5, H_6, H_9\}$ respectively.

The chain complex of X in Figure 4.8 is $\mathbb{Z}^3 \xrightarrow{\delta_2} \mathbb{Z}^{15} \xrightarrow{\delta_1} \mathbb{Z}^{13}$, where

$$\delta_2 = \begin{pmatrix} -1 & -1 & 0 & -1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & -1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & -1 & 1 & 1 & 1 \end{pmatrix},$$

$$\delta_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

This gives that $H_0(X) = \mathbb{Z}$ and all other homologies are trivial.

Definition 4.10. A *chain complex $C_*(K)$ of a pure permutahedral complex K* is a sequence of free abelian groups $C_0, C_1, \ldots$ together with homomorphisms $\delta_n : C_n \to C_{n-1}$, where C_i is the free abelian given by an ordered list of the i-cells and δ_i given by the boundaries of the i-cells.

The system of labelling k-cells as intersections of n-cells lends itself to a convenient description of a simplicial complex W which is the dual complex of a pure permutahedral complex K, whence we can calculate the homology of K.

- Let each facet in K represent a vertex of W.
- For every intersection of k facets present in K, let there be one k-simplex in W connecting the corresponding $k+1$ vertices.

This natural map gives rise to a homomorphism between the n^{th} homology groups of K and W. There exist many efficient algorithms for computing the homology of simplicial complexes, so we can easily find $H_n(W)$.

5 Implementation

HAP is a homological algebra library created by Ellis for use with the GAP computer algebra system. As part of this thesis we developed some theory and functions pertaining to cubical complexes as part of HAP, and an extension called HAP_Permutahedral which implements many of the same functions for permutahedral complexes. This chapter describes how the complexes defined in the previous chapters are created and represented in HAP and HAP_Permutahedral and describes the algorithms for functions which operate on such complexes. The code for HAP_Permutahedral can be found in Appendix C.

5.1 Representation and Creation

Recall Definition 2.13 In all cases for a complex K we define the index set

$$\Lambda(K) = \{\texttt{Cartesian}(\texttt{List}([1..\texttt{arraySize}(K)[i]]), i \in [1..\texttt{dimension}(K)\}.$$

The notation a_λ is taken to mean the entry at $A\texttt{!.binaryArray}[\lambda_1][\lambda_2]\ldots[\lambda_d]$.

The cases we have implemented are

- pure cubical complexes; where $\underline{T}$ is a list of orthonormal basis vectors,
- pure permutahedral complexes; where $\underline{T}$ is a list of d basis vectors $t_i = \{1, 1, \ldots, -d, \ldots, 1\}$ where the $(i+1)^{th}$ entry in t_i is $-d$.

Example 5.1. The 3-dimensional balls used in the pure cubical and pure permutahedral complexes, respectively, are

- $[[-1,-1],[-1,0],[-1,1],[0,-1],[0,0],[0,1],[1,-1],[1,0],[1,1]]$ and
- $[[-1,-1],[-1,0],[0,-1],[0,0],[0,1],[1,0],[1,1]]$.

A pure B-complex K corresponding to a space X is represented in HAP by a component object having three components:

- `K!.binaryArray` is a binary array; K`!.binaryArray` with entries (k_λ) for $\lambda \in \Lambda(K)$ where
$$k_{\lambda_1,\lambda_2,\ldots,\lambda_d} = \begin{cases} 1 & \text{if} \quad D(\lambda_1 t_1 + \ldots + \lambda_d t_d) \text{ lies in } X \\ \\ 0 & \text{otherwise} \end{cases}$$
- `K!.properties`: a list of pairs such as `["arraySize",[100,200]]`,`["IsContracted","true"]`
- `K!.ball`: a list which, when added to any entry k_λ in K gives all the neighbours of k_λ.

A *binary array* of dimension 1 is a list $L = [l_1, \ldots, l_n]$ where each l_i is either 0 or 1. The *array size* of L is the singleton $[n]$. A *binary array* of dimension $d \geq 1$ is a list $L = [l_1, \ldots, l_n]$ where each l_i is a binary array of dimension $d-1$ and all l_i have the same array size. The *array size* of L is the list of integers got by appending n to the array size of any l_i.

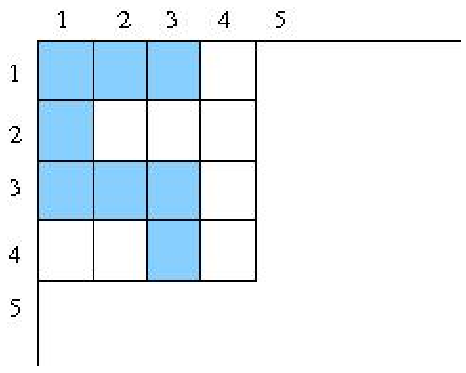


Figure 5.1: $X \subset \mathbb{R}^n_C$.

The pure cubical complex K where $X = |K|$ in Figure 5.1 has binary array

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}.$$

The following functions work for regularly tessellated complexes in general, though our implementations are directed at those with permutahedral and cubical tessellations. Note that in the more general setting of regular CW-spaces, it is not necessary to record a distinct data type for these objects - only the *ball*, which indicates the neighbourhood of a cell in the space, is recorded. The function `Dimension` returns the dimension of a pure complex, and the function `Size`, when applied to a pure complex K returns the sum of all k_λ ie. the number of facets in $X = |K|$. A pure complex K is obtainable from a binary matrix A using the functions `PureComplex` by creating a component object K, assigning K`!.binaryArray` to be A, and calculating K`!.properties Dimension` and `arraySize` from A.
A pure complex can also be obtained from a non-binary array A with threshold integer t using `ArrayToPureComplex`. Here all entries of A with value less than t are given the value '1' in K`!.binaryArray` and are 0 otherwise. The function `ReadImageAsPureComplex` reads a 2-dimensional image as an array whose entries are the RGB values of the pixels and uses a user-inputted threshold to thus create a pure complex. Similarly, `ViewPureComplex` converts the binary array from a pure complex to a black and white image.
The `View` function can also be used on three objects to provide an image of a 2-dimensional slice of the object. The function `View` inputs a d-dimensional pure complex and an integer list of length d. This integer list must have two zero entries, which indicate which perspective the image is being viewed from. Take, for example, the 3-dimensional object W, which consists of a stack of 20 of the 2-dimensional slices in Figure 5.2.

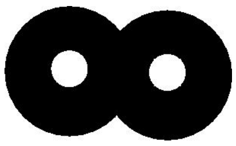


Figure 5.2: The 2-dimensional slice which we stack to form our three dimensional space W.

Using the `View` function in HAP_Permutahedral we can produce a sequence of 2-dimensional images of W from different perspectives.

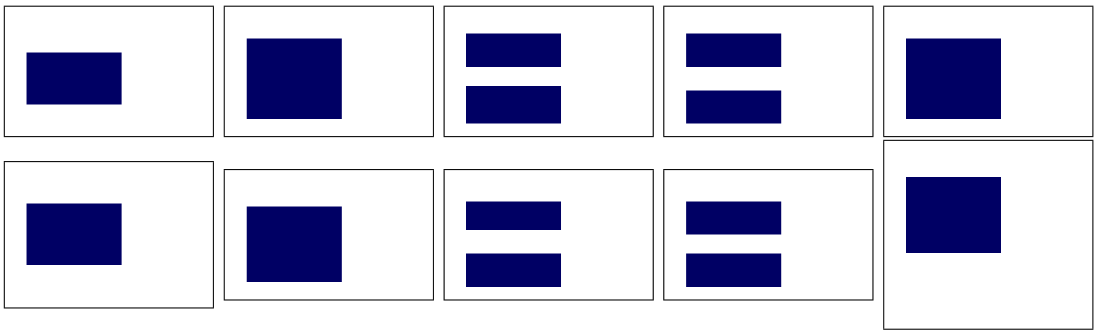


Figure 5.3: A selection of 2-dimensional slices viewed as though we were progressing through the object from left to right through the stack.

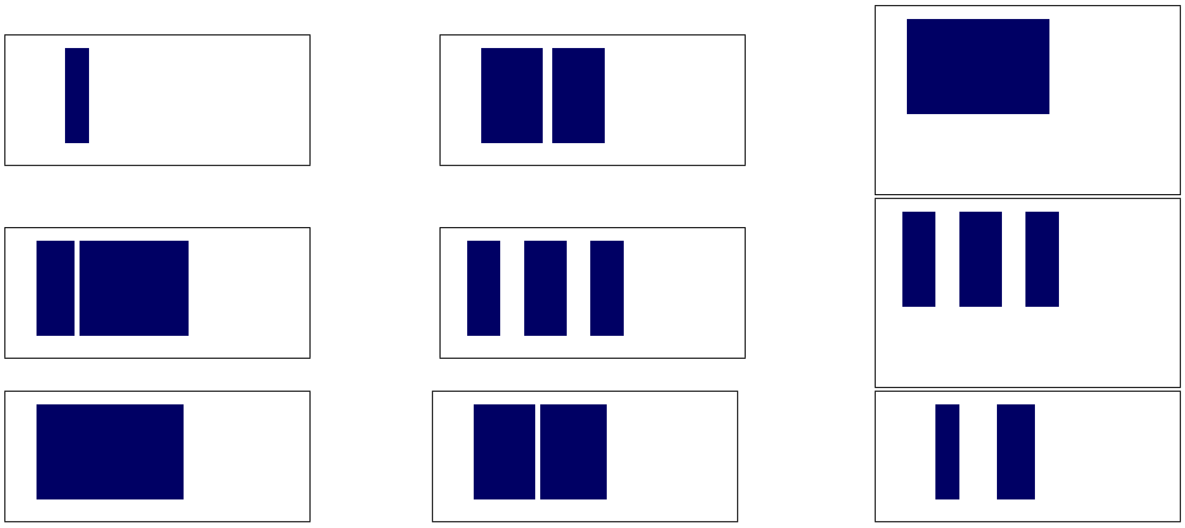


Figure 5.4: A selection of 2-dimensional slices viewed as though we were progressing through the object from the top down.

4-dimensional objects can be similarly viewed, though it is difficult to glean any meaningful information from them.

The function `Sample` inputs a pure permutahedral complex P and a positive integer n as arguments. It returns a subcomplex of P which is the union of n distinct randomly selected points.

The function `ShrinkPureComplex` can be useful in the case where we have a very large complex which might otherwise be difficult to work with. It "shrinks" the inputted pure permutahedral complex to a smaller, more manageable, pure permutahedral complex. This may cause the loss of some information, but in general the returned complex should offer a reasonable representation of the original.

Algorithm 5.2. ShrinkPureComplex

Input: An n-dimensional pure complex K

Output: A pure complex K' which is approximately homeomorphic to K, but whose binary array is only $\frac{1}{3^n}^{th}$ of the size.

Procedure:

```
Set $A :=$ EvaluateProperty($K$, "array size");
Set $K'$ to be a binary array with all zero entries and array size $[\frac{a_1}{3}, \frac{a_2}{3}, \ldots, \frac{a_n}{3}]$;
for $k'_\lambda \in \Lambda(K')$ do
  for $l \in \{-1, 0, 1\}^n$ do
    if $\sum k_{2\lambda+l} \geq \frac{3^n}{2}$ then
      Set $k'_\lambda = 1$;
    end if
  end for
end for
```

Another function which may be used to reduce the size of the complex is `CropPure-Complex`, which inputs a pure complex K and returns a pure complex L obtained from K by removing any "zero boundary sheets" of the binary array. Thus L and K are isometric as Euclidean spaces but there may be fewer zero entries in the binary array for L.

The function `BoundingPureComplex` inputs a pure complex K and returns a contractible pure complex L containing K.

The function `ComplementOfPureComplex` when applied to some pure complex K returns a copy K' of K where for every entry k with value 1 in K`!.binaryArray` the corresponding value k' in K'`!.binaryArray` is 0, and k' is 1 otherwise.

Algorithm 5.3. ComplementOfPureComplex

Input: A pure complex K

Output: A pure complex K' such that $k'_\lambda = 1$ if $k_\lambda = 0$ and $k'_\lambda = 0$ if $k_\lambda = 1$.

Procedure:

```
$K' := Copy(K)$;
for $k_\lambda = 1$ do
  $k'_\lambda := 0$;
end for
for $k_\lambda = 0$ do
  $k'_\lambda := 1$;
end for
```

A number of functions take two pure complexes (with the same array size) as their argument.

Algorithm 5.4. Pure Complex Union

Input: Two pure complexes I, J

Output: A pure complex K such that every k_λ is the maximal value of i_λ and j_λ.

Procedure:

```
$K := Copy(I)$;
for $k_\lambda = 0$ do
  if $j_\lambda = 1$ then
    $k_\lambda := 1$;
  end if
end for
```

Using similar algorithms, `PureComplexIntersection`(I, J) returns a pure complex K with binary array such that $k_\lambda = 1$ if and only if both $i_\lambda = 1$ and $j_\lambda = 1$, and `PureComplexD-ifference`(I, J) returns a pure complex K with binary array such that $k_\lambda = 1$ if and only if $i_\lambda = 1$ and $j_\lambda = 0$.

5.2 Neighbourhoods

The "d-ball" in relation to a d-dimensional pure complex K is defined to be the list of neighbours of a d-dimensional cell centred at the origin. This list can then be added to any cell f in the complex to give $N_K(f)$.
In the cubical case the d-ball for a cell with coordinates $[i_1, i_2, \ldots, i_d]$ consists of all $[i_1, i_2, \ldots, i_d]+\kappa$, where $\kappa \in [-1, 0, 1]^d$. For example, the neighbours of a 2-dimensional cell centred at the origin are $[[-1,-1],[-1,0],[-1,1],[0,-1],[0,0],[0,1],[1,-1],[1,0],[1,1]]$ so adding this d-ball to any cell f with coordinates $[i,j]$, its neighbourhood consists of $[[i-1,j-1],[i-1,j],[i-1,j+1],[i,j-1],[i,j],[i,j+1],[i+1,j-1],[i+1,j],[i+1,j+1]]$.

The d-ball for a d-dimensional pure permutahedral cell with coordinates $[i_1, i_2, \ldots, i_d]$ consists of all $[i_1, i_2, \ldots, i_d] \pm \kappa$, where $\kappa \in [0,1]^d$. For example, the neighbours of a 2-dimensional cell centred at the origin are $[[-1,-1],[-1,0],[0,-1],[0,0],[0,1],[1,0],[1,1]]$ so adding this d-ball to any cell f with coordinates $[i,j]$, its neighbourhood consists of $[[i-1,j-1],[i-1,j],[i,j-1],[i,j],[i,j+1],[i+1,j],[i+1,j+1]]$.

For all of the functions for pure complexes, unless otherwise stated, the only real difference in the implementation is the choice of d-ball. Consider the following algorithm which makes use of this "d-ball".

Algorithm 5.5. Thickened Pure Complex
Input: Pure complex K
Output: A pure complex K' such that if $k_\lambda = 1$ then every e in $N_{K'}(k'_\lambda)$ has value 1.
Procedure:

```
$K' := K$; repeat:=true;
for $k_\lambda$ in $\Lambda(K)$ do
  for $e$ in $d$-ball do
    $k'_{\lambda+e} := 1$;
```

```
        end for
    end for
```

The function `BoundaryOfPureComplex` makes use of a combination of some of the previous functions.

Algorithm 5.6. Boundary Of Pure Complex

Input: Pure complex K

Output: A pure complex K' such that $k'_\lambda = 1$ if and only if not every $k_{\lambda+d} = 1$.

Procedure:

```
    $A :=$ Complement($K$); $B :=$ Thickened($K$);
    $K' :=$ PureComplexIntersection($A, B$);
```

The function `ExcisedPureComplexPair` inputs a pure complex K and a subcomplex L and returns a list containing K' which is a copy of the pure complex K but with the interior of L removed, and the boundary of L .

Algorithm 5.7. Excised Pure Complex Pair

Input: Pure complex K and subcomplex L

Output: A list containing the pure complex K' such that $k'_\lambda = 1$ if and only if $k_\lambda = 1$ and l_λ is not in the boundary of L, and the boundary of L.

Procedure:

```
    $B :=$ BoundaryOfPureComplex($L$);
    $D :=$ PureComplexDifference($L, B$);
    $K' :=$ PureComplexDifference($K, D$);
    return $[K', D]$
```

The d-ball is also used in splitting a pure complex into its distinct path components. It selects an 'uncoloured' cell, 'colours' it, and then recursively assigns the same colour to all cells in the neighbourhood of cells of that colour. Another uncoloured cell is then chosen and given a new colour, whereupon the process is repeated. When there are no uncoloured cells remaining, the pure complex whose cells are the cells of the n^{th} colour is the n^{th} path component. It contains a recursive inner function `ColourNeighbours` which colours the neighbouring facets in a complex of a given facet the same colour as the facet itself.

Algorithm 5.8. Colour Neighbours

Input: Pure complex K and a coordinate list λ such that $k_\lambda = 1$

Ouput: A pure complex K' such that every facet in the same path component as k_λ has the same value.

Procedure:

```
for $d$ in $d$-ball do
  if not $x_{\lambda+d} = 0$ then
    $x_{\lambda+d} := x_\lambda$
    ColourNeighbours$(X, \lambda + d)$
  end if
end for
```

Let A be the array

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

PC := `PathComponentOfPureCubicalComplex`(A) splits this binary array into its separate path components in an object stored with the array

$$PC := \begin{pmatrix} 2 & 2 & 0 & 0 & 3 \\ 0 & 0 & 0 & 3 & 3 \\ 4 & 4 & 0 & 0 & 0 \\ 4 & 0 & 4 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \end{pmatrix}.$$

The 2nd path component then, for example has binary array

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Algorithm 5.9. Path Component Of Pure Complex

Input: Pure complex K and an integer n

Output: The n^{th} path component K' of K. If $n = 0$ then the number of path components is returned.

Procedure:

```
Set $count$= 1;
while $k_\lambda$ is 1 do
  count:=count+1;
  Add $\lambda$ to $K$!.representatives;
  ColourNeighbours($K, k_\lambda$)
end while
if $n = 0$ then
  return count;
else
  $A$:=StructuralCopy($K$!.binaryArray) $* 0$
end if
for $k_\lambda = n - 1$ do
  $a_\lambda := 1$;
  return PureComplex($A$)
end for
```

As part of the `PathComponentOfPureComplex` function, the inputted complex P is assigned a P!`.representatives`, which is a list of cells $\{c_1, \ldots, c_p\}$ where p is the number of path components and no two c_i, c_j are in the same path component. This is of use when constructing an acyclic subcomplex, or calculating homology.

The function SingularitiesOfPureComplex inputs a d-dimensional pure complex K, and positive integers r and t. It returns a subcomplex K' of K containing the union of k'_λ such that the region within the sphere of radius r about k_λ is 'non-differentiable' according to the given threshold t.

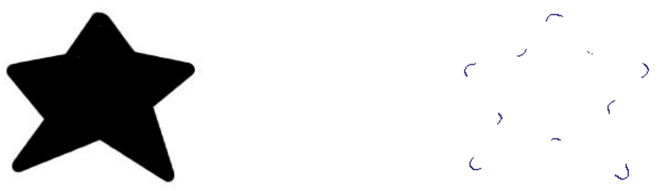


Figure 5.5: Singularities calculated with $r = 10$ and $t = 15$.

5.3 List of positive neighbourhoods

As mentioned in Chapter 2, a look-up list indicating the positive neighbourhoods in d-dimensional space can be created. Such a list is included for up to four dimensions in the permutahedral case in the package created as part of this thesis. A similar list for five dimensions would contain 4.6 trillion entries and fell beyond the scope of this project. The n^{th} entry in this look-up list for d-dimensional homotopy retractable neighbourhoods corresponds uniquely to a list whose entries are the digits in the binary representation of the integer n. The i^{th} entry of this binary list represents the neighbour, in some fixed position, of a facet f. If the n^{th} entry is 1 then the facet f can be removed without affecting the homotopy of the neighbourhood. Henceforth the "look-up value" of the neighbourhood of a facet k_λ is denoted by $\mathbb{L}(k_\lambda)$.

Use is made of this list in the following algorithm, which inputs a complex and a subcomplex thereof and returns a minimal subcomplex which is homotopy equivalent to the first complex subject to the condition that it contain the input subcomplex:

Algorithm 5.10. Homotopy Equivalent Minimal Pure Complex

Input: Pure complex K and subcomplex J.

Output: A minimal pure complex K' which is a homotopy retract of K and contains J.

Procedure:

```
$K' := K$; repeat:=true;
while repeat=true do
  repeat:=false;
  for $\lambda$ in $\Lambda(K)$ do
    if $k'_\lambda = 1$ and $j_\lambda = 0$ and $\mathbb{L}(k_\lambda) = 1$ then
      $k'_\lambda := 0$; repeat:=true;
    end if
  end for
end while
```

The function `HomotopyEquivalentMaximalPureComplex` operates similarly, but instead of removing cells, adds all possible cells to the subcomplex under the constraint that it must remain a subcomplex of K which is homotopy equivalent to J.

The function `ContractedPureComplex` uses `HomotopyEquivalentMinimalPureComplex`, but the subcomplex in the argument is empty, so all cells can be removed which do not change the homotopy of the original complex. `ContractibleSubcomplexOfPureComplex` uses the `Size` and `PathComponentOfPureComplex` functions to ascertain the largest path component of a complex, and then uses `HomotopyEquivalentMaximalPureComplex` to find a contractible subcomplex thereof. `AcyclicSubcomplexOfPureComplex` inputs a pure complex K and returns the maximal subcomplex K' of K such that $H_n(K') = 0$ for $n \geq 1$. These are useful for calculating the homology of a pure complex.

Using a combination of iterations of `CropComplex`, `BoundingComplex`, `HomotopyEquivalentMaximalPureSubcomplex` and `ContractedComplex` we can calculate the `ZigZagContractedComplex` of a pure complex.

5.3.1 Creating $\mathbb{L}_C^d$

To create the list $\mathbb{L}_C^d$ for the d-dimensional case a list of length 2^{3^d-1} was constructed, wherein the n^{th} entry corresponds uniquely to a list l of length $3^d - 1$ where the entries are digits of the binary representation of the integer n. Each entry in l refers to a fixed neighbour of a d-dimensional cell; the facet is present if and only if the corresponding entry in l is 1. If `Homology`$(N_K(f))$ = `Homology`$(\hat{N}_K(f))$ then the n^{th} entry in $\mathbb{L}_C^d := 1$. Calculating the homology of, in the three-dimensional cubical case, 2^{26} pure cubical complexes, would be time consuming, so the method used for verifying homology consistency is outlined here.
$N_K(f)$ is contractible for any facet f. Thus $H_0(N_K(f)) = 1$ and $H_n(N_K(f)) = 0$ for $n \geq 0$, and $\chi(N_K(f)) = 1$ for any f. So $N_K(f)$ is not positive if and only if

- $H_0(\hat{N}_K(f)) \neq 1$ or
- $H_n(\hat{N}_K(f)) > 0$ for some $n > 0$.

There can be no holes of dimension n in a d-dimensional space if $n \geq d$. To check for positivity in 2-dimensions it suffices to verify that the number of path components is 1 and that the Euler characteristic is 1, as $\chi(K) = \beta_0(K) - \beta_1(K)$ becomes $1 = 1 - \beta_1(\hat{N}_K(f))$, giving $\beta_1(\hat{N}_K(f)) = 0$.
In 3-dimensions, given Euler characteristic of 1 and one path component $1 = 1 - \beta_1(\hat{N}_K(f)) + \beta_2(\hat{N}_K(f))$.
The neighbourhood of an n-dimensional facet can contain at most one hole with $n-1$-dimensional boundary, which would require the neighbourhood to be homotopy equivalent to an $n-1$-sphere. In 3-dimensions the Euler Characteristic of an $n-1$-sphere is 2. Thus if $\chi(\hat{N}_K(f)) = 1$ and $\beta_0(\hat{N}_K(f)) = 1$ then $N_K(f)$ is positive.
This method does not hold for higher dimensions, but as the list $\mathbb{L}_C^4$ is of length $1.20892582 \times 10^{24}$ it would be impossible to store in any case. Using symmetries it might be possible to reduce the size of the list necessary, but this would require a number of extra checks to be

performed for every cell. This would negate the advantage of the efficiency of simply checking the corresponding entry in $\mathbb{L}_C^n$ to see whether the neighbourhood is positive or not.

5.3.2 Creating $\mathbb{L}_P^d$

As with the cubical case, to check for positivity of a facet f in 2 and 3 dimensions, it suffices to verify whether $\hat{N}_X(f)$ is path-connected and has Euler Characteristic 1. When creating the $\mathbb{L}_P^4$ however these conditions do not suffice. The n and $n-1$ homologies can be discounted as with the 2 and 3 dimensional cases, but it is possible that $\chi(\hat{N}_K(f)) = \beta_0(\hat{N}_K(f)) - \beta_1(\hat{N}_K(f)) + \beta_2(\hat{N}_K(f))$ is satisfied even though $\chi(\hat{N}_K(f)) = 1$ and $\beta_0(\hat{N}_K(f)) = 1$ and is not positive.

Consider the neighbourhood $N_A(f)$ of some facet f in A, with binary array

$$\begin{array}{lll}
[[0,1,1],[0,1,1],[0,0,0]] & [[0,0,1],[0,1,0],[0,0,0]] & [[0,0,0],[0,0,0],[0,0,0]] \\
[[0,0,1],[0,1,0],[0,0,0]] & [[0,1,1],[1,1,0],[0,0,0]] & [[0,0,0],[1,0,0],[1,1,0]] \\
[[0,0,0],[0,0,0],[0,0,0]] & [[0,0,0],[1,1,0],[1,0,0]] & [[0,0,0],[1,1,0],[0,1,0]]
\end{array}$$

We find that $\beta_0(\hat{N}_A(f)) = 1, \beta_1(\hat{N}_A(f)) = 1, \beta_2(\hat{N}_A(f)) = 1, \beta_3(\hat{N}_A(f)) = 0, \beta_4(\hat{N}_A(f)) = 0$ and $\chi(\hat{N}_A(f))) = 1$, so the neighbourhood is not positive as f's removal would alter its 1st and 2nd homologies. The 'path-connectedness' and 'Euler characteristic of 1' conditions are satisfied though. Therefore in 4 dimensions it is necessary to impose a third condition for a neighbourhood to be positive. Testing that either $\beta_1(\hat{N}_A(f)) = 0$ or $\beta_2(\hat{N}_A(f)) = 0$ forces the other to be 0, and thus along with the previous two conditions is sufficient to test for positivity. Since the homology computation of the neighbourhood of a cell in the permutahedral case is relatively slow, owing to the size of the boundary homomorphisms, we instead calculated the first homology of homotopically equivalent simplicial complexes. The implementation for this in GAP is more efficient, and as there are 2^{30} 'once-off' iterations, it makes sense to use this method. Owing to the symmetric nature of the construction, we only need to calculate and store half of the entries in the list.

5.4 Chain complexes

5.4.1 Cubical complexes

In HAP a cubical complex K with $X = |K|$ is represented as a component object consisting of

- a binary array X`!.binaryArray` $= (a_{\lambda_1,\lambda_2,\ldots,\lambda_n})$ where

$$a_{\lambda_1,\lambda_2,\ldots,\lambda_n} = \begin{cases} 1 \text{ if } & D(\lambda_1 t_1 + \ldots + \lambda_n t_n) \text{ lies in } X \\ \\ 0 & \text{otherwise} \end{cases}$$

- X`!.properties`: a list of pairs such as `["arraySize",[100,200],["IsContracted","true"]`

While this representation is similar to the representation of a pure cubical complex, it is important to note that they represent different objects. A '1' in a d-dimensional pure cubical complex X at x_λ represents a d-dimensional cell, whereas a '1' in a cubical complex at x_λ represents an m-dimensional cell, where m is the number of even entries in λ.
A pure cubical complex can be converted to a cubical complex as illustrated in section 3.2. Recall that $\xi(\lambda)$ is the number of even entries in λ. The chain complex of a cubical complex can be calculated using the following algorithm:

Algorithm 5.11. ChainComplexOfCubicalComplex
Input: A d-dimensional cubical complex X.
Output: Associated chain complex of X.
Procedure:

```
for every $x_\lambda$ in $[1..d]$ do
    Add $\lambda$ to a list $l_{\xi(\lambda)}$ of all $\xi(\lambda)$-dimensional cells.
end for
for $m$ in $[0..d-1]$ do
    Create an array of dimensions Length($l_{m+1}$) $\times$ Length($l_m$).
end for
```

```
for every $\lambda$ in $l_{m+1}$ do
  for every $\kappa$ in $l_m$ do
    if $\kappa = \lambda$ with ‘1’ added to one of $\lambda$’s odd entries then
      Add 1 to the $m + 1^{th}$ matrix
    end if
    if $\kappa = \lambda$ with ‘-1’ added to one of $\lambda$’s odd entries then
      Add -1 to the $m + 1^{th}$ matrix
    end if
  end for
end for
Set Dimension to be an internal function which, given an argument $n$ returns
the length of the $l_m$ list.
Set Boundary to be an internal function which, given arguments $i, j$ returns the
$j^{th}$ row of the $i^{th}$ matrix.
```

The homology, Euler Characteristic, Betti numbers, etc. can be calculated for a cubical complex using its chain complex, or of a pure cubical complex via conversion to a cubical complex.

5.4.2 Pure permutahedral complex

Unlike for pure cubical complexes, a pure permutahedral complex has no convenient corresponding “permutahedral complex”. Instead, we describe an n-dimensional cell in a d-dimensional pure permutahedral complex as the intersection of $d + 1 - n$ d-dimensional cells. The boundary of an m-dimensional cell which is the intersection of d-dimensional cells $p_1, p_2, \ldots, p_{d+1-m}$ d-dimensional cells consists of the $(m-1)$-dimensional cells which are the intersections of $p_1, p_2, \ldots, p_{d+1-m}, p_k$, where p_k is a neighbour of each d-dimensional cell $p_1, p_2, \ldots, p_{d+1-m}$. Therefore we use the function `PurePermutahedralComplex_Cells` which inputs an n-dimensional pure permutahedral complex and returns a list of length $n+1$ where the i^{th} entry is the list of $n+1-i$ intersecting facets representing the $i-1$-dimensional cells.

The GAP code extract below shows an example of this function.

```
gap> A:=[[0,1,1],[1,0,1],[1,1,0]];;
gap> P:=PurePermutahedralComplex(A);;
gap> C:=PurePermutahedralComplex_Cells(P);;
gap> C[3];
[[[2,3]],[[2,4]],[[3,2]],[[3,4]],[[4,2]],[[4,3]]]
```

(These are the 6 facets of P.)

```
gap> C[2];
[[[1,3],[2,3]], [[1,4],[2,3]], [[1,4],[2,4]],
[[1,5],[2,4]], [[2,2],[2,3]], [[2,2],[3,2]],
[[2,3],[2,4]], [[2,3],[3,2]], [[2,3],[3,3]],
[[2,4],[2,5]], [[2,4],[3,3]], [[2,4],[3,4]],
[[2,5],[3,4]], [[3,1],[3,2]], [[3,2],[3,3]],
[[3,2],[4,1]], [[3,2],[4,2]], [[3,3],[3,4]],
[[3,3],[4,2]], [[3,3],[4,3]], [[3,4],[3,5]],
[[3,4],[4,3]], [[3,4],[4,4]], [[4,1],[4,2]],
[[4,2],[4,3]], [[4,2],[5,1]], [[4,2],[5,2]],
[[4,3],[4,4]], [[4,3],[5,2]], [[4,3],[5,3]]]
```

(These are the 30 edges in P, each formed by the intersection of two facets.)

```
gap> C[1];
[[[1,3],[1,4],[2,3]], [[1,3],[2,2],[2,3]],
[[1,4],[1,5],[2,4]], [[1,4],[2,3],[2,4]],
[[1,5],[2,4],[2,5]], [[2,2],[2,3],[3,2]],
[[2,2],[3,1],[3,2]], [[2,3],[2,4],[3,3]],
[[2,3],[3,2],[3,3]], [[2,4],[2,5],[3,4]],
[[2,4],[3,3],[3,4]], [[2,5],[3,4],[3,5]],
[[3,1],[3,2],[4,1]], [[3,2],[3,3],[4,2]],
[[3,2],[4,1],[4,2]], [[3,3],[3,4],[4,3]],
[[3,3],[4,2],[4,3]], [[3,4],[3,5],[4,4]],
[[3,4],[4,3],[4,4]], [[4,1],[4,2],[5,1]],
[[4,2],[4,3],[5,2]], [[4,2],[5,1],[5,2]],
[[4,3],[4,4],[5,3]], [[4,3],[5,2],[5,3]]]
```

(These are the 24 vertices in P, each formed by the intersection of three facets.)

Algorithm 5.12. ChainComplexOfPurePermutahedralComplex

Input: A d-dimensional pure permutahedral complex K.

Output: Associated chain complex of K.

Procedure:

```
Cells:=PurePermutahedralComplex_Cells(K).
Set Dimension to be a function which, given an argument $n$, returns the length
of the list Cells[$n+1$].
Set Boundary to be a function which, given arguments $i, j$ returns the list of
orientations of the $i-1$-dimensional cells in the $j^{th}$ $i$-dimensional cell.
```

5.4.3 Chain complex of pair

For both pure cubical and pure permutahedral complexes, one can make use of chain complexes of excised pairs using functions such as the previously outlined `HomotopyEquivalentMinimalPureComplex` and `ContractibleSubcomplex`. Given a complex, X, one can choose a contractible subcomplex A and excise A from X using Algorithm 5.7, as described in Chapter 2.1, leaving a complex whose chain complex may be substantially smaller and easier to compute. See Example 2.9 in Chapter 1. The initial space had 4446 facets, but we can excise a contractible subcomplex thereof containing 4279 facets, enabling us to efficiently calculate the homology of the complex.

The Euler characteristic, Bettinumbers, and Homology can all be calculated from the chain complex.

5.5 Computation times

Here we give the timings for a number of computations to compare the permutahedral and cubical implementations. The complex K used throughout is from the image shown in Figure 5.6. All times are given in milliseconds and are the average of a number of computations. The "Size" listed is the number of facets in the resultant complex. The machine used is a 2GHz Linux laptop with 1Gb of RAM.

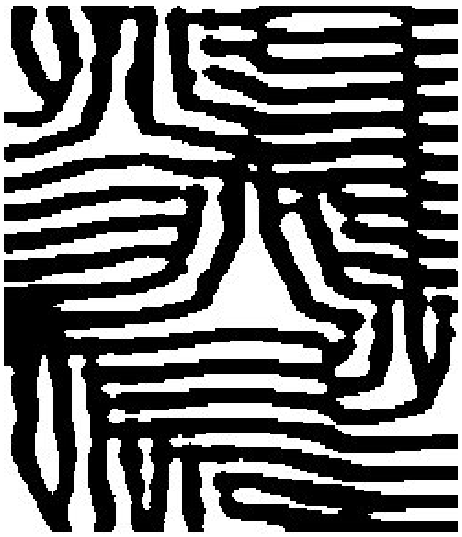


Figure 5.6: $X = |K|$

[2]

Table 5.1: Comparison of timings and resultant complexes

Function	Permutahedral timing	Size	Cubical timing	Size
BoundaryOfPureComplex	424	8933	364	9218
Complement	176	29940	180	29940
ContractedComplex	392	4303	360	2502
C:=ContractibleSubcomplex	976	36452	748	36451
HomotopyEquivalent-				
MaximalPureSubcomplex(K,C)	48	36452	60	36451
PathComponentOfComplex(K,2)	24	36968	28	36968
Singularities(K,10,15)	70640	3055	50567	3202
Thickened(K)	40	49372	52	49880
ZigZagContracted(K)	3780	334	4004	199

Table 5.2: Comparison of timings

Function	Permutahedral	Cubical
ChainComplexOfPair(K,C)	11317	1608
EulerCharacteristic(K)	5884	4636
Homology(K)	1828	1620

The following tables are iterations in 3-dimensions - in these cases the complex is P, where $Y = |P|$ is constructed by "stacking" ten copies of the previous X on top of one another.

```
P:=PureComplex(List([1..10],i->StructuralCopy(K!.binaryArray)));;
```

Table 5.3: Comparison of timings and resultant complexes

Function	Permutahedral	Size	Cubical	Size
BoundaryOfPureComplex(P)	9761	91590	11649	92180
Complement(P)	2937	299400	3408	299400
C:=ContractedComplex(P)	4980	4303	7061	2502
D:=ContractibleSubcomplex(P)	24862	366753	25537	366710
HomotopyEquivalent-				
MaximalPureSubcomplex(P,D)	944	366753	2196	366710
PathComponentOfComplex(P,2)	11337	369680	11813	369680
Singularities(C,5,10)	136148	4303	37635	2502
Thickened(P)	2452	498292	14657	498800
ZigZagContracted(P)	27538	339	20285	848

Table 5.4: Comparison of timings for chain complexes of permutahedral and cubical representations

Function	Permutahedral	Cubical
Homology	28386	34443

Taking X to be a further "stack" of 10 copies of Y, we calculate the homology of P where $|P| = X$. Note that we convert the pure permutahedral complex to a simplicial complex after contraction.

```
gap> L:=List([1..10],i->StructuralCopy(P!.binaryArray));;
gap> P:=PurePermutahedralComplex(L);
Pure Permutahedral Complex of dimension 4.
gap> SizeOfPurePermutahedralComplex(P);
4094900
gap> C:=ContractedComplex(P);time;
Pure Permutahedral Complex of dimension 4.
98186
gap> SizeOfPurePermutahedralComplex(C);
4303
gap> K:=PureComplexToSimplicialComplex(C,5);
Simplicial complex of dimension 2.
gap> Homology(K,0);time;
3
16
gap> Homology(K,1);time;
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
82493
gap> Homology(K,2);time;
[ ]
115943
gap> Homology(K,3);time;
[ ]
107851
gap> Homology(K,4);time;
[ ]
115723
```

5.6 Software Comparison

There are a number of related software packages which deal with the computation of homology and persistent homology such as

- CAPD [30] from the CHomP [2] group
- Plex [10].
- Dionysus [29]
- Kenzo [35]

There is some difficulty in comparing the efficiency of HAP with other software packages, as they serve different purposes and accept different inputs. Plex and Dionysus, for example, focus on the computation of persistent homology [1] based on implementing efficient row-reduction algorithms for the boundary matrices of simplicial complexes, whereas our aim was to implement efficient and modular software for general computations in homology to be incorporated into the GAP system. The approach used in HAP and HAP_Permutahedral also underlies the CAPD homology software [30] based on cubical complexes. As with Kenzo, our approach gives a central computational role to homotopy retracts, but our implementations are aimed more at data drawn from Euclidean spaces than from simply connected spaces arising in theoretical algebraic topology, which is the primary focus of Kenzo's package. HAP and HAP_Permutahedral aim to implement the general functionality of elementary algebraic topology in GAP, a pre-existing system for computational discrete algebra. Using HAP_Permutahedral and its algorithms implemented for permutahedral complexes, it is possible to efficiently compute the homology of 4-dimensional data sets. HAP works for tessellated spaces with lattice structure in general, with the sole difference in implementation for many algorithms between different tessellations being the adjustment of the neighbourhood of a cell, which is noted as a component of the object.

5.7 Abstract cubical complexes

A cubical complex $X \subset \mathbb{I}^N$ can be represented abstractly as the set $S = \{1, \ldots, 2^N\}$ of vertices of $\mathbb{I}^n$ together with a finite collection K of pairs of subsets (σ_k, τ_k) where $\{\tau_k \subset \sigma_k\}$ and τ_k, σ_k are subsets of S. In the poset of all subsets of S the interval from τ_k to σ_k has geometric realization a cube of dimension $|\sigma_k| - |\tau_k|$ if $\tau_k \subset \sigma_k$. To represent a pure finite cubical complex

$X \subset \mathbb{I}^N$ it is necessary and sufficient for all set differences $\sigma_k \backslash \tau_k$ to have common size. Any cubical complex can be expressed as an abstract cubical complex as outlined in Chapter 3.

Algorithm 5.13. Chain Complex Of Abstract Cubical Complex

Input: An abstract cubical complex X of dimension d.

Output: The chain complex of X.

Procedure:

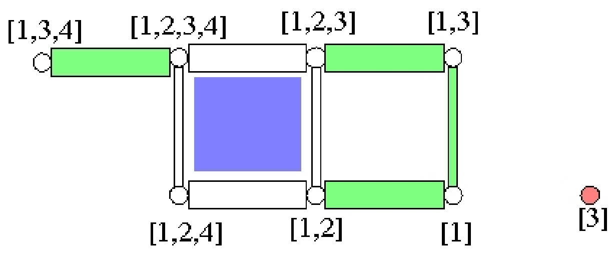
```
Separate the pairs in $X$ into lists $L_m$ of all $m$-dimensional pairs
for $m$ in $1..d$ do
  Create an array with Length($L_m$) rows and Length($L_{m-1}$) columns
  for every $m$-dimensional pair $m_i$ do
    Add the orientation of each $m-1$ dimensional pair $m-1_j$ to the $m^{th}$ array
    in the $j^{th}$ entry in the $i^{th}$ row
  end for
end for
Set Dimension to be a function which, given an argument $n$ returns the length
of the $L_n$ list.
Set Boundary to be a function which, given arguments $i, j$ returns the $j^{th}$ row
of the $i^{th}$ matrix.
```

The homology, Betti numbers, and Euler Characteristic of an abstract cubical complex can be calculated from the chain complex.

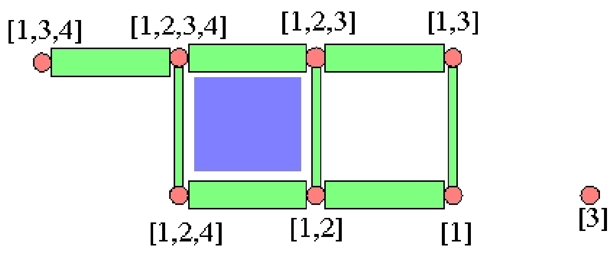
As well as calculating an abstract cubical complex from a cubical complex, we can also create an abstract cubical complex from a set of pairs P. Each $P = [P_1, P_2]$ is of the form $P_2 \subset P_1$. A set of pairs P admits the structure of an abstract cubical complex if and only if for every $[a, b]$ and $b \subset c$, $[a, c]$ is also present. The function `PairsToAbstractCubicalComplex` inputs a set of pairs and returns the abstract cubical complex which they form if they satisfy the above condition and returns `fail` and a missing pair if they do not. The function `MakePairsToAbstractCubicalComplex` inputs a set of pairs and then adds the extra pairs necessary to complete the abstract cubical complex.

```
gap> A:=[[[1,2,3,4],[1,2]],[[1,2,3],[1,3]],[[1,3],[1]],
[[1,2],[1]],[[3],[3]],[[1,2,3,4],[1,3,4]]];;
gap> P:=PairsToAbstractCubicalComplex(A);
[[1,2,3,4],[1,2,3,4]] must be a pair.
fail
gap> M:=MakePairsToAbstractCubicalComplex(A);
Abstract cubical complex of dimension 3.
gap> C:=ChainComplexOfAbstractCubicalComplex(M);;
gap> Homology(C,0);
[0,0]
gap> Homology(C,1);
[0]
gap> M!.pairs;
[[[[3],[3]], [[1,2,3,4],[1,2,3,4]], [[1,2],[1,2]],
[[1,2,3],[1,2,3]], [[1,3],[1,3]], [[1],[1]],
[[1,3,4],[1,3,4]], [[1,2,4],[1,2,4]]],
[[[1,2,3],[1,3]], [[1,3],[1]], [[1,2],[1]],
[[1,2,3,4],[1,3,4]], [[1,2,3,4],[1,2,3]],
[[1,2,3],[1,2]], [[1,2,3,4],[1,2,4]], [[1,2,4],[1,2]]],
[[[1,2,3,4],[1,2]]], []]
```

In Figure 5.7 (a) is illustrated a possible geometric realisation of the abstract cubical complex with cells as defined by A above, with 2-cells in blue, 1-cells in green, and 0-cells in red. Applying `PairsToAbstractCubicalComplex` returns fail and informs us of one of the missing cells which needs to be filled in. `MakePairsToAbstractCubicalComplex` bypasses this and simply fills in the necessary cells, leaving us with (b).



(a) `A:=[[[1,2,3,4],[1,2]],[[1,2,3],[1,3]],[[1,3],[1]],[[1,2],[1]],[[3],[3]],[[1,2,3,4],[1,3,4]]]`



(b) `gap> M:=MakePairsToAbstractCubicalComplex(A);`

Figure 5.7: Using `MakePairsToAbstractCubicalComplex` to force a selection of pairs to be part of an abstract cubical complex.

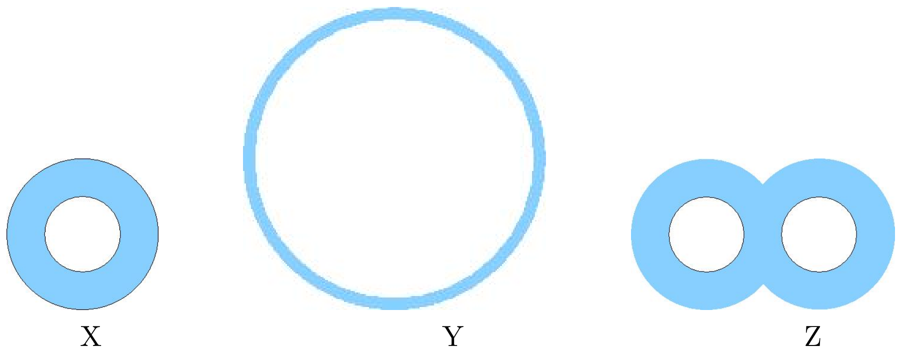
6 Potential Applications

6.1 Estimating Topology

It was suggested in Chapter 2 that we could estimate the homology of some topological space X by calculating the persistent homology of a random sample $S \subset X$ of the population through a number of thickenings. Table 6.1 below lends credibility to this idea by illustrating that for a range of standard mathematical spaces $X \subset \mathbb{R}^n$ we can recover the topology of X by thickening a small sample of points from X.

- Let X denote an annulus in the plane with outer radius 2 and inner radius 1. We consider X as a population from which we wish to sample. We represent X as a pure cubical complex involving 9575 2-cells. Its first three Betti numbers are 1,1,0.
- Let Y denote an annulus in the plane with outer radius 10 and inner radius 9. We consider Y as another population from which we wish to sample. We represent Y as a pure cubical complex involving 9472 2-cells. Its first three Betti numbers are 1,1,0.
- The population Z is equal to the union of X and X' where X' is obtained by translating X through a distance 3. It contains 18866 2-cells. Its first three Betti numbers are 1,2,0.

- The population W is a hollow cube obtained by removing a solid cube of side 5 from the centre of a solid cube of size 13. Its first three Betti numbers are 1,0,1.
- We can also consider direct products $M \times N = \{(m, n) : m \in M, n \in N\}$. If M and N



are pure cubical complexes of dimensions a and b respectively, then $M \times N$ is naturally a pure cubical complex of dimension $a + b$.

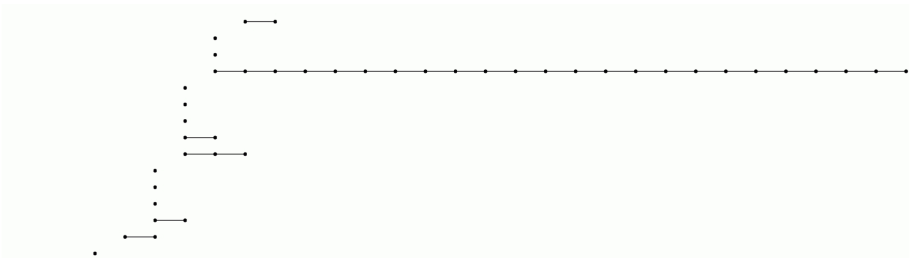
- Let T be a hollow square obtained by removing a solid cube of side 14 from the centre of a solid cube of size 20. We represent T as a pure cubical complex. Let the population $U = T \times T$. The population U then has the homotopy type of a torus, with 121584 4-cells and Betti numbers 1,2,1.

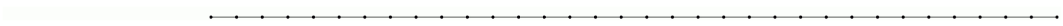
- Our samples are finite sets of points chosen randomly from the population according to a uniform distribution. We deem a sample S to be *representative* of a population M if the Betti numbers of M agree with those of a significant number of consecutive thickenings of S. We consider this number of thickenings to be significant if it is greater than or equal to a quarter of the number of thickenings required to turn S into an object with one path component which will not yield any holes through further thickening. The accuracy column gives the percentage of respresentative samples among ten selections of S. For efficiency in the case where $M = U$ we use discrete Morse theory and compare the number of critical cells of the population M and the thickenings of the sample S respectively, as opposed to the Betti numbers.

Table 6.1: Estimating population topology from samples

Population	Sample size	Accuracy
X	429 (5%)	100%
X	96 (1%)	100%
X	48 (0.5%)	90%
X	24 (0.25%)	90%
X	10 (0.1%)	40%
Y	473 (5%)	100%
Y	95 (1%)	100%
Y	48 (0.5%)	100%
Y	24 (0.25%)	100%
Y	9 (0.1%)	50%
Z	943 (1%)	100%
Z	566 (0.75%)	80%
Z	377 (0.5%)	50%
Z	189 (0.25%)	40%
W	384 (20%)	90%
W	192 (10%)	60%
W	96 (5%)	60%
W	19 (1%)	0%
U	384 (1%)	100%
U	192 (0.75%)	60%
U	96 (0.5%)	20%

In Figure 6.1 are a few examples of persistent homology bar codes for some of the spaces used above.


(a) The β_1 persistent homology of a series of thickenings of $S \subset X$ with 96 cells.


(b) The β_1 persistent homology of a series of thickenings of $S \subset Y$ with 95 cells.

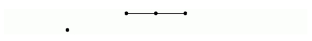

(c) The β_2 persistent homology of a series of thickenings of $S \subset W$ with 142 cells.

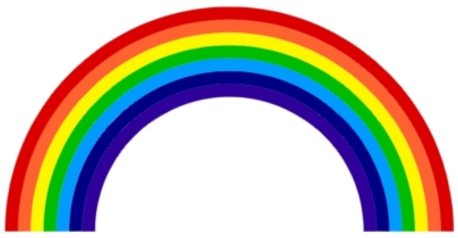
Figure 6.1: A selection of bar codes from the above samples.

6.2 Image segmentation

Image segmentation can be described as the process of partitioning a digital image into multiple segments. [38] The aim of segmentation is to change an image's representation into something that is more meaningful or easier to analyse. With image segmentation, every pixel in an image will have certain common visual characteristics, such as colour, intensity or structure. In Figure 6.2 we see an image divided by means of colour. This was done by varying the input threshold value in our `ReadImage` algorithm, which calculates the sum of the RGB values of a pixel and compares it to a user-input threshold.

The main method of image segmentation which we are interested in is feature recognition. Using functions such as the `Boundary` and `Singularities` detailed in Section 2.7, we aim to detect the edges and vertices of images. Ideally, detecting the edges in an image would lead to curves which indicate the boundaries or discontinuities of objects. In this manner we can filter out some 'less relevant' information, whilst still preserving the important structural information. The examples used are two-dimensional for ease of illustration, but the methods can equally be applied to higher dimensional data.

It is not always possible to obtain ideal edges from real life images of even moderate complexity - *fragmentation* is an issue; where edge curves are not connected, and we also


(a) Original image "Rainbow"

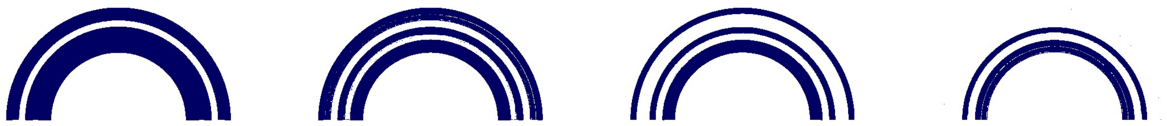

(b) "Rainbow" converted to a pure cubical complex with thresholds of 480, 405, 300 and 210 respectively.

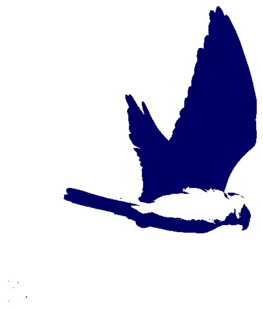
Figure 6.2: Segmenting an image by colour.

encounter *false edges*; where 'unimportant' edges are highlighted.

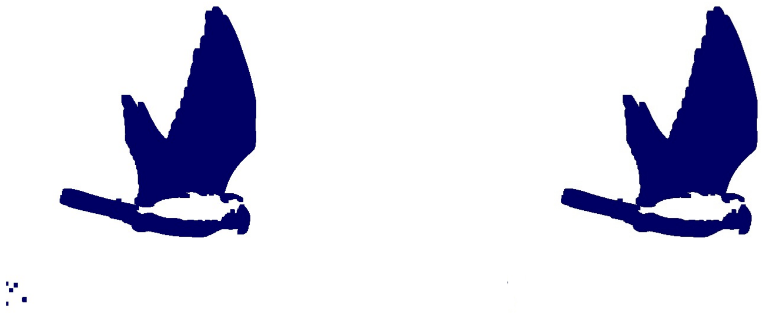
Fragmentation can sometimes be countered by thickening, as in Figure 6.3 *(c)*, but this is not always the case. Likewise, false edges, which crop up regularly in medical imaging or as 'noise', or outliers in statistical data, can sometimes be 'cleaned up' by removing path components of less than a certain size as in Figure 6.3 *(d)* where we created a 'copy' of the original image which only includes those path components greater than a certain size. Our software's handling of the aforementioned errors in complicated images is rather crude, and there is better imaging software available for this, but we include these examples here to illustrate that our algorithms, with some amendments, could be used independently for image analysis.

In Figure 6.3 [45] we convert the photograph into a black and white image representing a pure cubical complex. Some fragmentation is evident, as the kite appears to be in two parts. After a couple of thickenings to amend this, we notice some false edges in the form of a number of small dots to the left of the picture, which register as extra path components. Simply removing all path components of less than a given area gets rid of these. Taking the boundary of the resulting image gives an object with significantly fewer data points (2496, as opposed to 53754) which nevertheless provides a good approximation to the original. Calculating the singularities of this and thickening then gives us five path components, which correspond to the wing tips, the tail, the beak, and the 'crossing' of the wings.

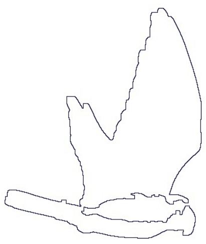
(a) A photograph of a kite.



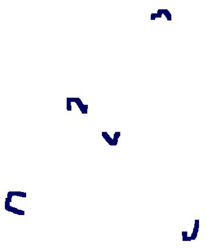
(b) The image as converted to a pure cubical complex.



(c) A number of thickenings ensures the kite is a single path component, and then we 'clean' the image up by removing any very small path components.



(d) We can take the boundary of this to give a faithful representation which lends itself to easier computation.



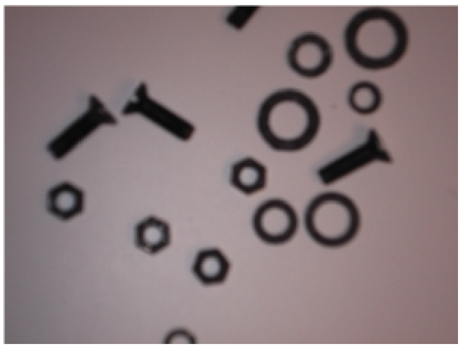
(e) The singularities of the image are located at the wing-tips, tail-tip, beak, and wing-crossing of the bird.

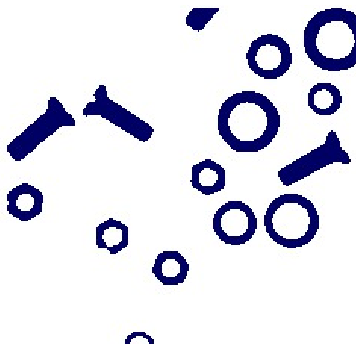
Figure 6.3: Singularity detection.

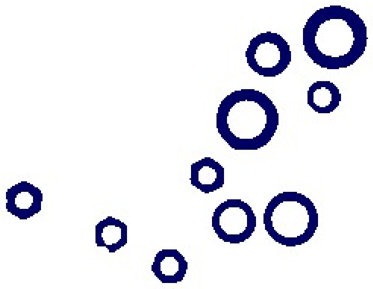
Suppose we want to calculate the number of washers, nuts and bolts in Figure 6.4 The number of bolts can easily be calculated as the number of path components less the number of one-dimensional holes. But as both nuts and washers are single path components with one-dimensional holes, we must then take note of a components singularities. We note that the boundary of a washer seems smooth, whereas the boundary of a washer contains singularities at six points. Finally, we can distinguish between washers of different sizes using thickenings or size analysis.

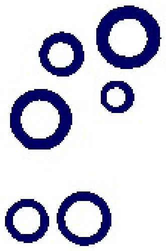
The code used to generate the images is shown here.

```
gap> A:=ReadImageAsPureCubicalComplex("washersnutsbolts.jpg",250);;
gap> ViewPureCubicalComplex(A)
(b)
gap> B:=PureCubicalComplex(A!.binaryArray*0);;
gap> for i in [1..PathComponentOfPureCubicalComplex(A,0)] do
> if not EulerCharacteristic(PathComponent(A,i))=0 then
> B:=PureCubicalComplexUnion(B,PathComponentOfPureCubicalComplex(A,i));;
fi;od;
(c)
gap> C:=PureCubicalComplex(B!.binaryArray*0);;
gap> for i in [1..PathComponentOfPureCubicalComplex(B,0)] do
> if Size(SingularitiesOfPureCubicalComplex(PathComponent(B,i),5,50)=0 then
> C:=PureCubicalComplexUnion(C,PathComponentOfPureCubicalComplex(B,i));;
fi;od;
(d)
gap> D:=PureCubicalComplex(C!.binaryArray*0);;
gap> for i in [1..PathComponentOfPureCubicalComplex(C,0)] do
> p:=ThickenedPureCubicalComplex(PathComponentOfPureCubicalComplex(C,i)
> for j in [1..7] do p:=ThickenedPureCubicalComplex(p);;
> if EulerCharacteristic(p)=0 then
> D:=PureCubicalComplexUnion(D,PathComponentOfPureCubicalComplex(C,i));;
fi;od;od;
```


(a) A photograph of nuts, washers and bolts


(b) The photograph in *(a)* represented as a pure cubical space X.


(c) We remove from X all path components with no 1-dimensional holes


(d) We then remove any path components which have singularities above a certain tolerance. This removes the "nuts".

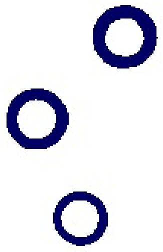

(e) We can then discern between larger and smaller washers by assessing how many thickenings a path component permits before its 1-dimensional hole disappears. Here, we have removed the smaller "washers".

Figure 6.4: Discerning objects from one another using segmentation.

This example contains no overlapping objects, though it is conceivable that such a situation might occur. Using combinations of previously mentioned techniques, it is possible, in many cases, to distinguish the overlapping objects within a component. A component comprised of overlapping bolts, for example, will have Euler characteristic 1. A component comprised of a pair of overlapping washers or nuts will have Euler characteristic 0 or -1 or -2, in which case we might analyse the components' singularities to find out which the objects are. It is, of course, possible to create a scenario where such two-dimensional analysis will not work. Two washers stacked directly on top of one another will be virtually indistinguishable from a single washer without using three-dimensional data, for example.

Another geometric method for distinguishing objects is to use the function `PuncturedPurePermutahedralComplex`, which inputs a pure permutahedral complex and integer r and creates an empty sphere of radius r about the centre of gravity of the pure permutahedral complex. In Figure 6.5 none of the objects have useful singularities, but by analysing a sequence of 'punctures' of the pure permutahedral complex, we can see that two of the objects will have two path components which persist for a number of rs. This helps us to distinguish between long, thin objects and short, fat ones, for example, which might have the same homology and non-distinguishing singularities.

The code used to generate the images is shown here.

```
gap> A:=ReadImageAsPureCubicalComplex("punctured.jpg",300);;
(a) gap> B:=PurePermutahedralComplex(StructuralCopy(A!.binaryArray*0));;
gap> for r in [10,50,80] do
> for i in [1..PathComponentOfPurePermutahedralComplex(A,0)] do
> B:=PurePermutahedralComplexUnion(
> B,PuncturedPathComponentOfPurePermutahedralComplex
> (PathComponentOfPurePermutahedralComplex(A,i),r));; od;
> ViewPurePermutahedralComplex(B); od;
```

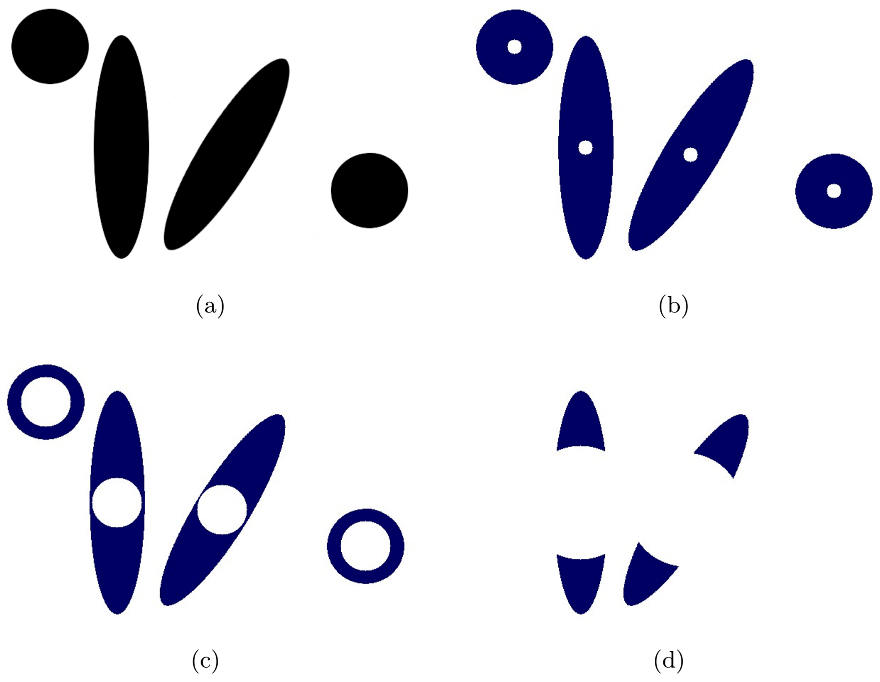
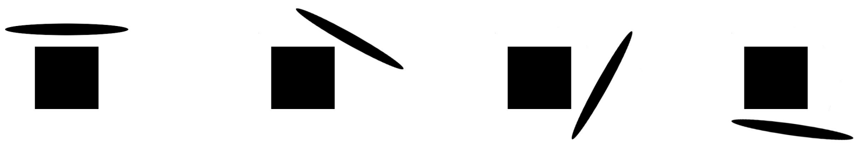


Figure 6.5: Two of the objects have two persistent path components for a range of radii.

6.3 Tracking Components

One of the original motivating examples behind our research was the analysis of MRI scans, specifically, the tracking of the movement of particular organs etc. within the body. If δ is the length of time or distance, for example, between successively created images of an n-dimensional space X containing non-intersecting components $c_1..c_t$ whose movement is continuous, then for some sufficiently small δ it is possible to embed the sequence of X_is into an $(n+1)$-dimensional space and use PathComponent to identify a given path component at each stage.

A 3-dimensional MRI scan of a patient is a sequence of hundreds of 2-dimensional images. While a doctor must visually identify which component in a particular slice corresponds to a given organ or body part, it would seem practical and beneficial to automate the process of then identifying this same body part of interest in the hundreds of other 2-dimensional slices. While it transpired that obtaining data from medical images was impractical, a toy example illustrating the idea is given in Figure 6.6. Given more specialised software, it should theoretically be possible to monitor the movement of specific organs etc. within the body.



A series of images of a space X with two path components.

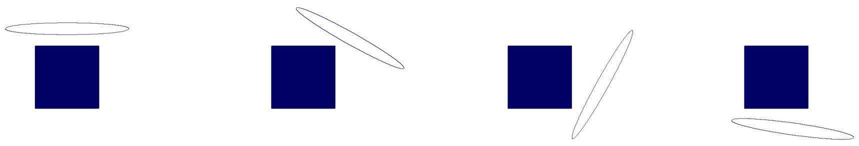
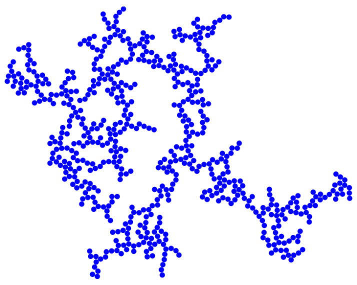


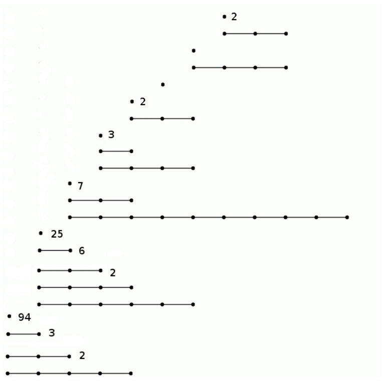
Figure 6.6: We track the first path component as it moves through the space X.

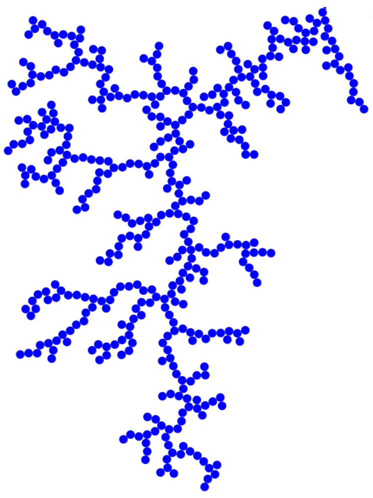
6.4 Measuring Shape

In their paper, *"Measuring Shape with Topology"*, McPherson and Schweinhart [28] "propose a measure of shape which is appropriate for the study of a complicated geometric structure, defined using the topology of neighborhoods of the structure". They analyse three examples of polymers constructed by different processes "which are all models for physical processes of interest" and the main component of their analysis relies on calculating the persistent homology of the polymers. To illustrate that our computer package could be useful for such analysis, we take a copy of a section of each of the polymers they use and calculate its first persistent homology through a number of thickenings. The times taken for the three examples over ten thickenings were 1616, 218, and 1036 seconds respectively. Further, using permutahedral complexes' property that $Cont(Comp(X) \cong Comp(Cont(X)$ it is sometimes easier to calculate $\beta_0(Comp(X))$ instead of $\beta_1(X)$.

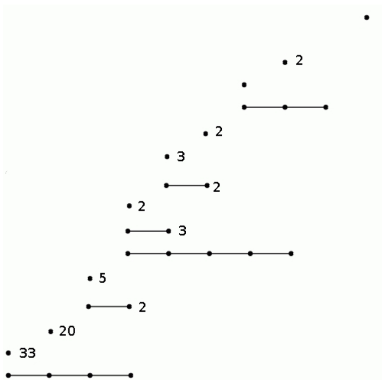


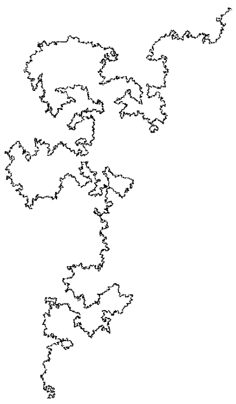
(a) Sample from a branched polymer. Source: McPherson & Schweinhart [28]



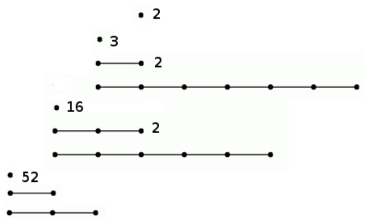


(b) Sample from a Brownian tree. Source: McPherson & Schweinhart [28]





(c) Sample from a self-avoiding walk. Source: McPherson & Schweinhart [28]



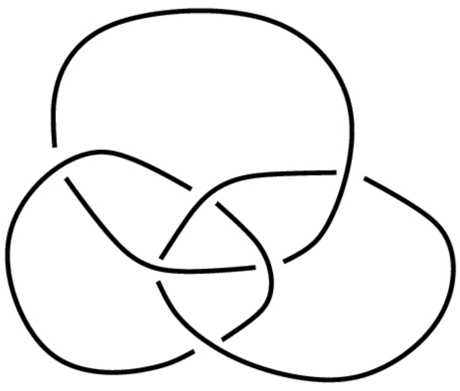


Figure 6.8: B: A link diagram for Borromean rings embedded in Euclidean 3-space.

6.5 Knots

HAP's functions for tessellated spaces can be used to obtain a presentation for the fundamental group of the complement of a link diagram. Working with permutahedral complexes, the property that two n-dimensional permutahedra 'touch' if and only if they share an $(n-1)$-cell, as detailed in Chapter 5, enables us to compute the critical cells of the complement of a link diagram more efficiently. Table 6.2 illustrates the computation of the critical cells of the Borromean rings in Figure 6.5. The function `ReadLinkImageAsPurePermutahedralComplex("image.jpg")` inputs an image file containing a link diagram and attempts to output the corresponding link as a 3-dimension pure permutahedral complex.

Table 6.2: Comparison of computation times to obtain a fundamental group

Cubical	**Size**	**Time**	**Permutahedral**	**Size**	**Time**
			Contracted(B)	3238	33
Crop(B)	33654	4	Crop		1
Complement	1025724	3	Complement	352642	1
Contracted	114556	259	Contracted	70716	30
ToCWComplex		63	ToCWComplex		32
CriticalCells		33	CriticalCells		13

The pure permutahedral complex method took 13 seconds as opposed to 33 seconds in the cubical case. The critical cells in the pure permutahedral case are $[[2,1],[2,635],[2,4164],[1,137211],[1,139265],[1,200276],[0,62659]]$. The code below provides information about the generators and relators of the fundamental group. The fundamental group of the complement of a link diagram is an invariant and might conceivably be used to help distinguish between complicated knots.

```
gap>M:=FundamentalGroupOfRegularCWComplex(L);
there are 3 generators and 2 relators of total length 22
<fp group of size infinity on the generators [ f1, f2, f3 ]>
gap> RelatorsOfFpGroup(M);
```

$[f1 * f3^-1 * f2^-1 * f3 * f2 * f1^-1 * f2^-1 * f3^-1 * f2 * f3, f1^-1 * f3^-1 * f2 * f1 * f3 * f1^-1 * f3^-1 * f2^-1 * f3^2 * f1 * f3^-1]$

Conclusion and Future Work

A topic of central interest in homological algebra is the computation of homological invariants of finite spaces. We have shown that certain computations performed using a permutahedral tessellation can be more efficient than those using cubical or simplicial complexes. The goal of this project was the development of efficient and practical routines in the computational algebra language GAP for the manipulation of topological datasets. To this end, we have produced a user-friendly extension to the software package HAP, complete with a manual, tutorial, code for functions which provide for both geometrical and homological manipulation and analysis, and comprehensive lists of the contractible neighbourhoods of cubical and permutahedral complexes for up to three and four-dimensional data respectively. Using the permutahedral complex we have been able to extend the capabilities of certain functions to include efficient implementations in four dimensions.

- Certain existing algorithms in our package could no doubt be made more efficient.
- Our implementation for calculating persistent homology relies on contractions of spaces. It is theoretically possible that these persistent homology computations could be implemented using zig-zag retractions instead which could speed up the process substantially.
- Our implementations focus primarily on cubical and permutahedral tessellations, but it would be feasible to adjust the operations to accept pure B-complexes in general, where the relevant neighbourhood would be required as an extra input variable.
- It is likely that the implementation of certain cohomology functions would be somewhat similar in nature to algorithms we have implemented.

- Although we have thus far failed to realise the initial aim of using our software for MRI analysis, we do have certain functions which could be of theoretical benefit if the data were interpretable. It might be worth pursuing the possibility of providing a compatability layer to synthesise our package with some existing MRI software for further investigation, and it would certainly be worth pursuing a slight expansion on the `ReadImage` function to enable it to accept ranges of differing RGB values which could then be used to analyse all data of one colour, for example.

Bibliography

[1] H. Adams, *JPlex with Beanshell tutorial* (2011) `comptop.stanford.edu/u/programs/jplex`

[2] M. Allili, Z. Arai, M. Gameiro, T. Kaczynski, W. Kalies, K. Mischaikow, M. Mrozek, P. Pilarczyk, T. Wanner, Computational Homology Project `http://chomp.rutgers.edu/software/`

[3] M. A. Armstrong, *Basic Topology* Springer (1979)

[4] D. Bailey, D. Townsend, P.Valk, M. Maisey *Positron Emission Tomography: Basic Sciences* Springer (2004)

[5] R. Bott, L. Tu *Differential Forms in Algebraic Topology* Springer (1982)

[6] G. Carlsson *Topology and data* Bulletin of the American Mathematical Society 46 (2009) 255-308

[7] G. Carlsson, A. Zomorodian, A. Collins, and L. Guibas *"Persistence barcodes for shapes"* Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry Processing (2004), 124-135

[8] M. Cohen, *A* course in simple homotopy theory, Graduate Texts in Maths 10, Springer-Verlag, (1973).

[9] J. Conway, N. Sloane *S*phere Packings, Lattices and Groups Grundlehren der Mathematische Wissenschaften 290 Springer (1999)

[10] G.Carlsson *et al.* PLEX, a software package for persistent homology of simplicial complexes, `comptop.stanford.edu/u/programs/jplex`

[11] G. Denham and P. Hanlon *On the Smith Normal form of the Varchenko bilinear form of a hyperplane arrangement.* Pacific Journal of Mathematics (1997), 123-146

[12] G. Ellis, *Algorithmic Homology lecture series*

[13] G. Ellis HAP - *Homological Algebra Programming* Version 1.10 (2012), a package for the GAP computational algebra system. `http://www.gap-system.org/Packages/hap.html`

[14] G. Ellis and F. Hegarty, *Computational homotopy of finite regular CW-spaces*

[15] H. Edelsbrunner, J. Harer, *Persistent homology - a survey*, in Twenty Years Afterwards AMS (2007)

[16] H. Edelsbrunner, J. Harer, *Computational Topology. An Introduction.* AMS (2010)

[17] D. Farley, *Finiteness and CAT(0) properties of diagram groups*, Topology (2003)

[18] R. Forman *Morse theory for cell complexes* Advances in Mathematics 134 (1998), 90-145

[19] R. Forman, *"A user's guide to discrete Morse theory"* Seminaire Lotharingien de Combinatoire 48 (2001)

[20] The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.4.12*; (2008), `(http://www.gap-system.org)`.

[21] F. Gray *"Pulse Code Communication"* U.S.Patent 2,632,058 (1953)

[22] P. Grout *Truncated octahedron* .png file created using Blender

[23] M. Hall, *Theory of Groups* American Mathematical Society, Chelsea, (1976)

[24] S. Harker, K. Mischkaikow, M. Mrozek, V. Nanda, H. Wagner, M. Juda, P. Dlotko *The Efficiency of a Homology Algorithm based on Discrete Morse Theory and Coreductions* Proceedings of the 3rd International Workshop on Computational Topology in Image Context (2010), 41-47

[25] A. Hatcher, *Algebraic Topology.* Cambridge University Press, Cambridge, (2002)

[26] T. Kaczynski, K. Mischaikow, M. Mrozek *Computational Homology* Springer (2004), pp. 480

[27] W.S. Massey *A Basic Course in Algebraic Topology*, Graduate Texts in Mathematics 127, Springer-Verlag (1991)

[28] R. McPherson, B. Schweinhart, *Measuring Shape with Topology* arXiv:1011.2258v2 [math.AT] (2010)

[29] D. Morozov *et al.* Dionysus, a C++ library for computing persistent homology `www.mrzv.org/software/dionysus`

[30] M. Mrozek *et al.*, Computer Assisted Proofs in Dynamics software, `capd.ii.uj.edu.pl`

[31] G. Muellehner, J. Karp, A. Guvenis *A method for reconstructing images from data obtained with a hexagonal bar positron camera.* Transactions on Medical Imaging (1985)

[32] G. Muellehner, J.G.Colsher, R.M.Lewitt *A hexagonal bar positron camera: Problems and Solutions* Transactions on Nuclear Science (1983)

[33] J. R. Munkres, *Elements of Algebraic Topology.* Perseus Books Pub., New York, (1994)

[34] A. Postnikov *Permutohedra, associahedra and beyond* International Mathematics Research Notices (2005)

[35] A. Romero, F. Sergeraert *Discrete Vector Fields and Fundamental Algebraic Topology* Preprint (2010) `www-fourier.ujf-grenoble.fr/~sergerar/Kenzo`

[36] J. Rotman, *An Introduction to Algebraic Topology* Springer (1988)

[37] J. Rotman, *An Introduction to Homological Algebra* Springer (2009)

[38] L. Shapiro, G. Stockman *Computer Vision* Prentice-Hall pp279-325 (2001)

[39] L. Soicher, *The Joy of GAP Packages* Proceedings from The Groups in Galway Workshop (2009)

[40] A. Tropsha, C.W. Carter, S. Crammer, I.I. Vaisman *Simplicial Neighbourhood Analysis of Protein Packing (SNAPP): A Computational Geometry Approach to Studying Proteins* Methods in Enzymology, Volume 374 (2003) `http://dx.doi.org/10.1016/S0076-6879(03)74022-1`

[41] Xiangjian H., Wenjing J., Qiang W., T. Hintz *Description of the cardiac movement using hexagonal image structures* Computerized Medical Imaging and Graphics 30 (2006), 377-382

[42] Xiangjian H., Wenjing J., Namho H., Qiang W., Jinwoong K., *Image Translation and Rotation on Hexagonal Structure* Sixth IEEE International Conference on Computer and Information Technology (CIT'06) (2006) pp.141

[43] H. Xiong *Digital Image Processing* `ivm.sjtu.edu.cn/files/dip/IVM_DIP_Lecture02.pdf`

[44] A. Zomorodian and G. Carlsson, *"Computing persistent homology"* Discrete Computational Geometry 33, (2005), 249-274

[45] `http://www.public-domain-image.com/cache/fauna-animals-public-domain\`
`\-images-pictures/birds-public-domain-images-pictures/kittiwake-birds\`
`\-pictures/white-tailed-kite-bird-in-flight-elanus-leucurus_w579_h725.jpg`