Algorithms for Sustainable System Topologies

Tihana Galinac Grbac$^{[0000-0002-4351-4082]}$, Emili Puh, and Neven Grbac$^{[0000-0001-6657-6297]}$

Juraj Dobrila University of Pula, Zagrebačka 30, HR-52100 Pula, Croatia
`{tgalinac,epuh,neven.grbac}@unipu.hr`

Abstract. These are the follow-up lecture notes of the lectures presented at the SusTrainable Summer School 2022 held at the University of Rijeka, Croatia, in July 2022. The main goal of the lectures is to provide a gentle introduction to topological data analysis because of its possible applications to sustainable software engineering. When applying topological data analysis to software system structures, one can observe additional structural metrics from the topological space which may be useful as a complement analysis technique in understanding complex software system behavior while aiming to address required quality attributes and sustainable goals.
Specific challenges on addressing sustainability as software quality attribute, but also as an aspect of engineering software, are discussed. A running example is presented and how the topological data analysis can be used to analyse software structures is explained, related implementation details are discussed, and these observations are related to sustainable software and software engineering practice. The topological notions and ideas are introduced and explained on a toy example, in which the basic topological invariants are calculated explicitly, and then an exercise is provided in which the students can test their understanding and check the efficiency of their algorithms on examples of complex software structures.

Keywords: Software systems · Software structure · Topological data analysis · Computational topology · Algebraic topology.

1 Introduction

Digitalization is nowadays recognized as a strategy for driving business success and a key strategy for implementing global sustainable goals in various domains. The software is the main tool used to abstract and model the physical world driving digitalization in all aspects of human life and as such it becomes a crucial target in addressing global sustainability goals. Therefore, sustainability has become one of the global goals for further technological evolution [18, 19].

Adequate modeling of software logic and algorithms is important to address sustainability goals from different perspectives. **Engineering sustainable software** targets optimization of computation in big data analysis in applications such as medicine and environmental modeling cases, optimization of physical resource usage targeting performance or energy efficiency constraints supporting decision-making processes in various applications such as business processes, traffic management, communal infrastructure management, etc. and addressing reliability, security and responsibility aspects of software operation in various mission-critical domains. To support these tasks, the software implements various advanced machine learning and artificial intelligence algorithms that may operate on big data, may be distributed across the entire telecommunication network, and may require extensive processing power and related resources. Here the main problem is the effective and efficient use of computational power and resources. In the software engineering domain, **sustainable software engineering** is a twofold act, it considers theory and practice that support software engineering tasks developing software that would secure sustainable software operation within the target application domain as mentioned above, but also to support sustainable business goals for the software engineering industry like for example to support long-term software evolution and responsible profession towards all stakeholders involved in the software lifecycle value chain. Sustainability goals significantly reflect on the software engineering profession, and all traditional software engineering methods need additional reassessment from this new perspective. Some examples are code optimization, code analysis, detecting anomalies, bug tracking and resolution, support in communication and social networking among stakeholders engaged in the software lifecycle, support in predictive maintenance, and software modeling oriented to long-term software evolution.

In all these software applications the key problem is the management of complexity. In our previous lecture notes [13, 14], we define **complex systems** as systems in which it is hard or impossible to derive simple rules of global software behavior from local system behaviors. Global system behavior may be characterized through the system properties measured as a consequence of the software system execution. Examples of global system properties are defined by ISO Quality model [20]. For example, in software systems, we may measure reliability as the software system quality property of being able to work or operate for long periods without breaking down or needing attention. Furthermore, there is work in progress to revise standard software quality attributes following global sustainability goal [32, 24, 23], in which we may measure sustainability as the global system property of the use of natural resources and energy in a way that does not harm the environment. Both global properties are sensitive to decisions we undertake during the system design phase while modeling software logic and algorithms. On the other hand, local software behavior may be characterized by dimensions of the local system properties. Examples of local system properties are various metrics that can be measured during the software and system engineering phases on the product, process, or even project level, like for instance, various cyclomatic complexity metrics on system components, component defectiveness, or effort spent on component development, people expertise, etc. As is defined in the above-stated definition of a complex system, the main problem in engineering such a complex system is the absence of simple models that can guide design decisions, e.g. how to develop system structure and its components' complexities that would lead to desired global system behavior during its execution and use in the real environments.

The *system structure* is the main artifact we model in any complex system design. The possible solution space in which we design components of complex systems and their interactions is extremely large. Proper design decisions can select system structures whose execution can achieve the best global properties during system execution. We may understand this concept of connecting two levels of system abstraction through the case of music. The music is designed through music composition and the composed music is felt as sonority during music reproduction, see Fig. 1 reused from [12]. Note that these two levels of music abstraction provide completely different views on music.

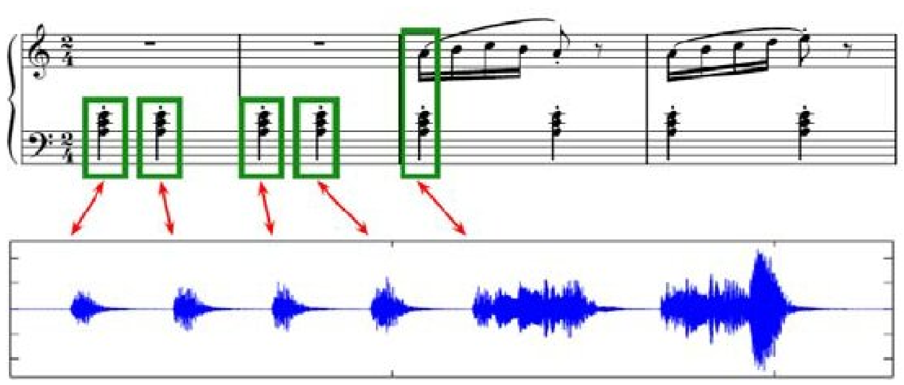


Fig. 1: Design time music composition view (top) versus sonority during music reproduction view (bottom). Image reused from [12].

Graphs are an essential and frequently used tool in various fields of engineering complex systems structures due to their ability to model complex systems by connecting global behaviors to local properties and their internal relationships in a comprehensible and manipulable manner. Numerous graph-based algorithms can be used to analyze and model system behavior effectively. However, the main problem of the majority of graph-based algorithms is computational efficiency and lack of structural property metrics. Topological Data Analysis (TDA) is a useful analysis method because of its ability to analyze shapes and structures within the topological space. Graphs used in the computing discipline may capture topological space of data where data can be any analyzed artifact from big data in various applications to the data coming from software structure. The use of TDA provides a unique perspective on the structural and dynamic properties of analyzed data, offering insights that complement traditional analysis methods. The main benefit of TDA is that it brings a new qualitative perspective on the analyzed system structure (or data structure) that may be related to other global and local system properties [25].

These lecture notes explain how we may apply TDA to engineering sustainable software. In our running case, we will show how TDA may be used in the analysis of software structures with the help of TDA aiming to address sustainable software evolution. The results of our case of using TDA to model software structures in evolution are presented in [37].

The main learning objectives of these lecture notes are as follows:

- To understand specific challenges of modeling and management of sustainable software systems
- To understand the needs and benefits of generalized approaches to software modeling and software management
- To introduce students with the key concepts from topological data analysis
- to learn how to implement abstract topological notions,
- to understand the difficulties in terms of computational power and sustainability of the algorithm by testing the code on different types of graphs,
- to understand how we can impact on sustainable software structures

The lecture notes are structured as follows. This introductory section is followed by Sect. 2 in which we define a playground for the application of TDA within software engineering context. Here the application of complex system definition is discussed in the context of a software system, the complex software system modeling and design problems and solutions are pointed out, and the landscape of sustainable software structures is defined. In Sect. 3, TDA is motivated and introduced. In Sect. 4, the examples of algorithms in TDA are presented. Sect. 5 explains the principles of TDA algorithms on a toy example. In Sect. 6, an exercise for students to practice the TDA principles on software structures is given. Finally in Sect. 7 we conclude the paper.

2 Sustainable software structures

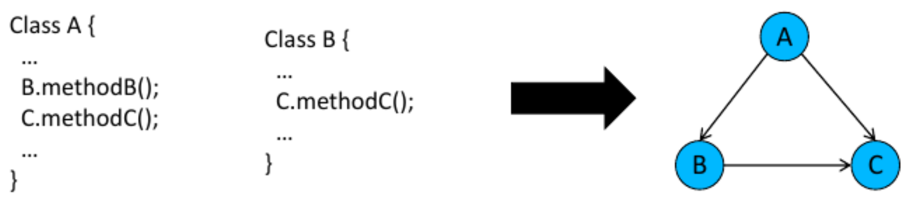
There are several specific challenges of the modeling and management of sustainable software systems.

Software engineering discipline addresses problems of system reuse, independent evolution of system parts, and development of generic software functions that are getting harder and more expensive as the complexity of software increases. The consequences of badly designed software solutions may be severe and affect operational software behavior, making software product operations and their development ineffective in achieving sustainable goals. Advanced software management tools are needed to enable smarter software creation thus fulfilling not only functional goals but also operational sustainability of resource management.

The specific challenge of designing complex systems is that it is hard to understand, manage, maintain, and evolve, and especially several people with diverse knowledge is needed to develop them. So, the responsible software engineering profession has to enable the independence of people working together on the same complex system, to enable its understanding for its easier maintenance and long-term evolution. The design principles used in the software engineering discipline that we introduced in [13, 14] are modular system designs aiming to develop the system as a set of loosely coupled components, and with a number of levels of abstraction with a layered design and clear hierarchy. Therefore, any (non-trivial) software system has modular structures and consists of modules (components) and their interactions. The term module refers to a component with the standard and loosely coupled interfaces that are used by other modules within its environment. Depending on the context and the purpose of the study, the modules may be subroutines, functions, units, classes, objects,...

Another challenge of modern software systems is that they usually operate in a globally interconnected Internet environment, modules may be distributed over the Internet network, modules may be invoked at runtime, and modules may be replaced at runtime. The system structure is then composed not only of many components that are run within isolated computer nodes, but the software structure is a set of software components that are interconnected on a geographically distributed Internet network. Decisions on software structure may and will also influence the sustainability of network operations. One of the key challenges that 6G networks should address is how to lower energy consumption [1]. This work is an attempt in that direction, since the majority of software-based resource management solutions are deployed via telecommunication networks and smart designs of software structures may lead to significant savings. These are specific challenges of modeling and management of sustainable software systems for distributed software deployment environments.

To address the aforespecified challenges the software engineering discipline has developed various generalized approaches to software modeling and management. One widely used abstract artifact is software structure. The software structure is defined as a set of modules and their interactions that are used to achieve global system functionality. The software structure is an abstract artifact and there are numerous ways to represent software structure. One of the most common software structure representations is by using call graphs [4].



```
Class A {
  ...
  B.methodB();
  C.methodC();
  ...
}
```

```
Class B {
  ...
  C.methodC();
  ...
}
```

Fig. 2: Call graph as a representation of software structure [34].

An example of representing Java code as a call graph is presented in [34] and the concept is reused from [38]. In this example, the graph is used for abstract software structure representation, in which program classes (A, B, C) are represented as graph nodes (vertices) and method calls (methodB(), methodC()) are represented as directed graph edges. The same principle we reused to develop a graph extraction tool *rFind* to extract call-graphs from Java code [34], which were analyzed with the help of motifs. Although, this analysis has provided us with useful insights into software structure evolution the main problem was in its computational inefficiency. Within this lecture, the students were encouraged to use the same call-graph structures to exercise with TDA described in Sect. 6. The results of the TDA analysis on the same call graphs were presented in [37] where the experiences of using motifs versus TDA were explained in detail.

Our previous studies analyzed software system behavior with the help of this abstract representation of software communication structure as a graph network, in which we experiment with network science models on such software graphs [39, 34, 33]. With the help of various structural characterization approaches, the aim is to understand the system behavior such as reliability as a global system property. These works are an extension of our previous studies in *Software Defect Prediction* [27] that aim to predict high-risk defective modules based on the models trained on historical data of local software metrics. The main challenge in these works is dimensionality reduction and feature selection procedures, [16, 31]. In the related survey studies, it has been identified that an improvement in SDP models could be gained with the additional characterization of the problem, not represented within the standard software metrics. These models suffer from a lack of system structure knowledge that may be incorporated into the data analysis and provide a more powerful basis for data clustering. TDA may be used for clustering of the graph structure by determining the extent to which nodes in a graph tend to form local clusters or groups that may improve the effectiveness and efficiency of decision-making algorithms. In our previous work, we have already demonstrated that software structure may contain useful information to model software defect prediction [39, 34]. Some hidden structural characterization obtained from topological information and higher dimensional structure can be obtained from the TDA models such as Betti numbers (which we explain in Sect. 4). These may help to better distinguish between differences and similarities during the data clustering phase. This topological information has been already identified as powerful tool in various data analysis domains [6, 26]. For example in healthcare and disease detection [22], neuroscience [21], image analysis and computer vision [2, 8, 3], protein classification [5], gene expression data classification [9], epidemiology [35], software failure dynamics [36], etc.

3 Topological Data Analysis

Topology is a branch of mathematics that deals with qualitative geometric information. This discipline studies geometric properties in a way that is much less sensitive to the actual choice of metric than simple geometric methods, which involve quantitative geometric properties such as distance, angle, area, or volume. More precisely, topology ignores the quantitative values of distance functions and replaces them with the notion of infinite proximity of a point to a subset in space. Roughly speaking, topology studies the shape and form of a geometric object and ignores its size and position.

This metric insensitivity is useful in studying situations in which one only understands the metric roughly. Topological data analysis refers to the study of data using topological methods. It is particularly useful to describe and compare the shape, discover trends, and search for hidden patterns in the data.

The standard reference for the basic general topology is the classic book by Munkres [30]. For the algebraic topology, the recommended references are [29] and [17]. There are several books and survey articles explaining the various applications of topological data analysis, such as [7], [6], [28], [11]. For an account of open problems in computational topology see [15].

3.1 Topology as a qualitative model of reality

Sometimes in the study of real-world behavior, it is not important to have precise quantitative models. In such cases, the interest is in the shape, trends or form, and not in size, scale or time frame. The models of reality are then qualitative, not quantitative. The qualitative models come in two types: discrete and continuous. Common examples of discrete models are graphs, which may be viewed as a set of vertices (also called nodes, points) together with a set of edges (also called branches). Edges are pairs of vertices, usually represented as a line connecting two vertices. Graphs may be used to model connections between objects, but without actual scale or size of them. Weighted graphs may be used if more quantitative information is required.

Topology is a part of geometry that deals with the qualitative properties of geometrical objects. It provides a model of the shape of a geometrical object, without referring to its size. Very roughly speaking, two objects are topologically equivalent if they can be transformed into each other by stretching and modeling but without gluing and cutting. A folklore example for topological equivalence is the example of a doughnut and a coffee mug in Fig. 3, because if they were made of clay one could model one to another without cutting and gluing. However, the pot in the figure is not topologically equivalent to the doughnut and the coffee mug. The reason is that the pot has two handles with a hole in each, and the doughnut and a coffee mug have just one hole. Hence, in order to get equivalence we must either glue one of the holes in the handles of the pot or cut a hole in the coffee mug, but both gluing and cutting is forbidden.

Topological space is a set of points for which a certain notion of “closeness” is defined. This is achieved by defining for each point a set of (open) neighborhoods. For example, the real line is a topological space in which the neighborhoods of each point are open intervals containing that point. Another example would be the plane. It is a topological space in which the neighborhoods of each point are open discs containing that point. These examples are shown in Fig. 4.

(a) Doughtnut

(b) Coffeemug

(c) Pot

Fig. 3: Example of topological equivalence.

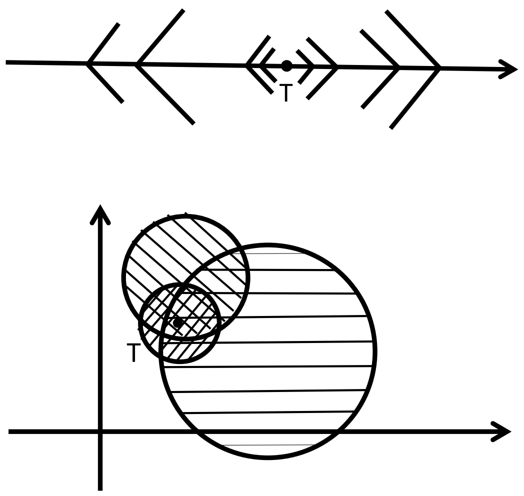


Fig. 4: Neighborhoods on a line and in the plane as examples of topological spaces.

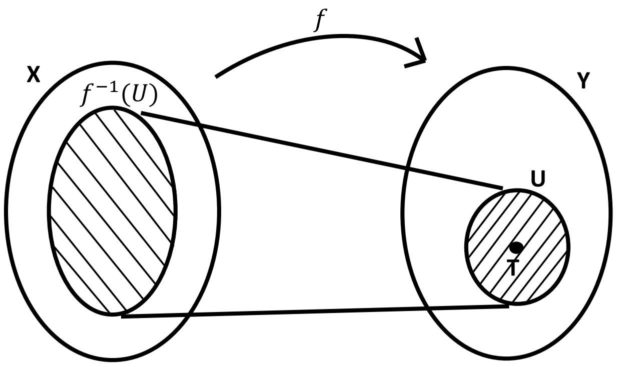


Fig. 5: Continuous functions between topological spaces preserve neighborhoods.

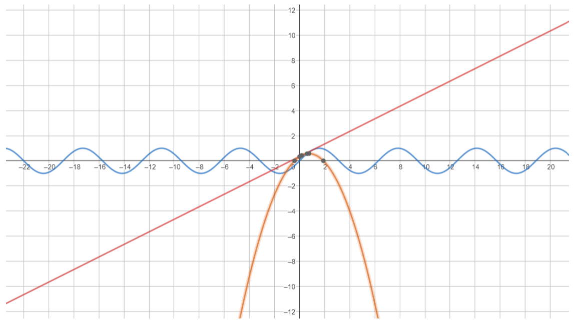


Fig. 6: Approximations of the sine function with a straight line and a parabola around the point $x = \frac{\pi}{3}$.

The topological spaces are compared using continuous maps, which in some sense preserve "closeness", which is encoded in neighborhoods. Under a continuous map, the preimage of an open neighborhood of each point is again an open neighborhood. A schematic image of this condition is given in Fig. 5. Homeomorphism is a continuous bijection between two topological spaces such that its inverse is also continuous. Two topological spaces are topologically equivalent if there is a homeomorphism between them. It is extremely difficult to check that two topological spaces are homeomorphic or, in other words, topologically equivalent. Often one cannot find or construct a homeomorphism between topological spaces, but also cannot exclude the possibility of its existence, so that the question of their equivalence cannot be settled.

Considering and comparing topological spaces is important in topological data analysis, as the topological space itself contains information on the underlying data, its structure and properties.

3.2 Topological invariants as approximations

Topological invariants are certain objects attached to a topological space, which remain unchanged under homeomorphisms, i.e., topologically equivalent spaces have the same topological invariants. Thus, if one finds any topological invariant of two topological spaces which is not the same, they are certainly not topologically equivalent, but still, if all topological invariants that can be explicitly computed or described of two topological spaces are the same, it is still not sufficient to decide whether they are topologically equivalent or not. There are many topological invariants, most of them of algebraic nature. Hence, the study of topological invariants is often referred to as algebraic topology. The most common invariants are the fundamental group, homotopy, homology and cohomology, and persistent homology.

We draw now an analogy to, hopefully, more familiar subject of mathematical analysis. In mathematical analysis, which is a very quantitative field of mathematics, the approximations of functions is one of the most useful tools in applications. Given, for instance, a real function on some interval, one may use derivatives to obtain the best approximation with a straight line of that function near a point (the so-called tangent line). Using higher order derivatives, the Taylor polynomials provide approximations with polynomials of higher degree. The example[1] of approximations of the sine function with a straight line (the Taylor polynomial of degree one) and a parabola (the Taylor polynomial of degree two) aroung point $x = \frac{\pi}{3}$ is given in Fig. 6. It is often in applications that only such approximation of the observed phenomenon is available (interpolation polynomials). Polynomials are elementary functions, which are more suitable for calculations than arbitrary functions, while at the same time provide a reasonable approximations of the real world phenomena.

Unlike mathematical analysis, topological spaces are not at all quantitative. The approximations of topological spaces should somehow approximate the shape and play the same role as polynomials do in mathematical analysis. (Algebraic) topological invariants may be viewed precisely as algebraic objects that approximate a topological space. They are simpler than the original topological space, but still resemble its shape often in highly sophisticated ways. Algebra is in some sense easier than topology.

3.3 From topological space to its invariants

The passage from a topological space to its algebraic topological invariants usually requires some kind of discrete model of a continuous object such as a topological space. This is achieved using the so-called triangulation. The topological space is described using a skeleton, which is similar to a graph describing the space, but contains also higher dimensional pieces (not only vertices and edges). Such skeleton is called a simplicial complex and it is a generalization of a graph.

A triangulation by a simplicial complex is a generalization of graphs in which also higher dimensional pieces, and not only vertices and edges, are present. These higher dimensional pieces are called simplices. In the spirit of graph theory, a d-dimensional simplex is simply a set of $d+1$ vertices (points), together with all lower-dimensional subsimplices, i.e., the non-empty subsets. Observe that an edge of a graph may be viewed as a set of two points. Thus, together with its

[1] More precisely, the function $f(x) = \sin x$ is approximated around the point $x = \frac{\pi}{3}$ by the straight line

$$T_1(x) = \frac{1}{2}x + \left(\frac{\sqrt{3}}{2} - \frac{\pi}{6}\right)$$

using the first derivative of f at $x = \frac{\pi}{3}$, and by the parabola

$$T_2(x) = -\frac{\sqrt{3}}{4}x^2 + \left(\frac{\pi\sqrt{3}}{6} + \frac{1}{2}\right)x + \left(\frac{\sqrt{3}}{2} - \frac{\pi}{6} - \frac{\pi^2\sqrt{3}}{36}\right)$$

using also the second derivative of f at $x = \frac{\pi}{3}$.

non-empty subsets, which are its end-points, it is an 1-simplex. Vertices of a graph are 0-dimensional simplices, as they can be viewed as sets with one point. More generally, a triangle, together with its sides and vertices, may be viewed as a 2-dimensional simplex. Similarly, a tetrahedron is a 3-dimensional simplex.

Any (finite) set of simplices, viewed as a single object, is a simplicial complex. The point is that any topological space may be triangulated by a simplicial complex, which may be viewed as a discretization of a topological space. The triangulation by a simplicial complex is not unique, but different triangulations lead to certain topological invariants which are equal. Hence, assignment of certain topological invariants to a topological space using a convenient discretization in terms of the simplicial complex really makes sense.

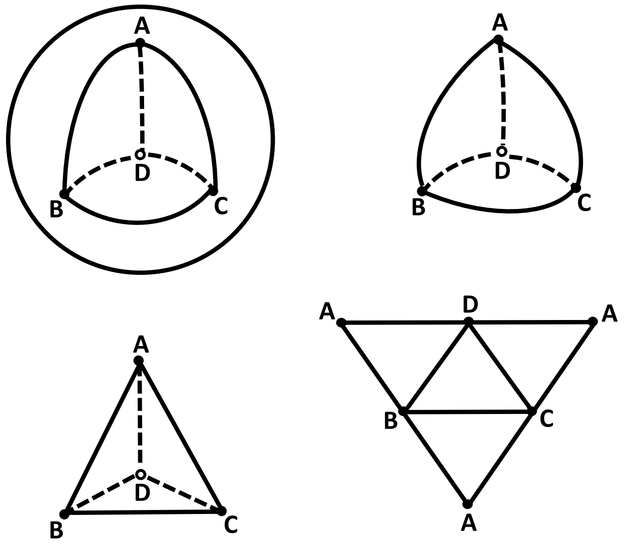


Fig. 7: A triangulation of a two-dimensional sphere.

Examples of triangulations are given in Fig. 7 and Fig. 8. The former shows a triangulation of a 2-sphere, and the latter the trinagulation of a 2-dimensional torus. A triangulation of a 2-sphere is presented on the sphere itself, extracted out of a sphere and with straight edges, but also as a plane diagram in which the points and segments with the same names are identified. In the case of a 2-dimensional torus, a triangulation is more complicated, so that only the points of the triangulation are shown in the figure, and not the full triangulation of the torus. It is given as a plane diagram instead, in which the points and segments with the same names are identified.

The relationship between the study of topological spaces and the study of their invariants can be nicely pointed out by comparing zoology and paleontol-

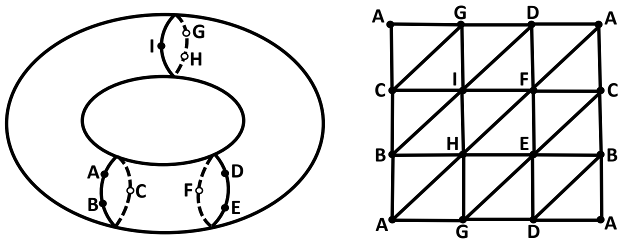


Fig. 8: A triangulation of a two-dimensional torus.

ogy. In paleontology the study of dinosaurs relies mostly on the animal skeletons, which can be compared to the study of simplicial complices and the associated algebraic objects. On the other hand, zoology can study animals as a whole, but also consider only the skeletons, which can be compared to the study of a topological space as it is given, but also the algebraic invariants can be used. However, it is quite often the case in topology that the topological spaces are so complicated that we are more in the skin of a paleontologist than a zoologist. Or, in analogy with the music example provided in Sect. 1 the software structure is like a music sheet structure where we explore its properties and try to understand its consequences on sonority in music waveforms during its reproduction. Again, the relation among various structural differences in sheet music and its audio representations are so complicated and we need advanced methods for their better understanding, [12].

3.4 Topological analysis of software

The study of software in terms of a graph is an old and widespread idea. It turns out to be very useful in all stages of the software life-cycle from the design, testing, and verification to maintenance.

Any (non-trivial) software system consists of components. Depending on the context and the purpose of the study, the components may be modules, software units, classes, objects, functions, and so on. The components may be viewed as vertices of a graph and the connections between components may be viewed as edges of a graph. These together form a graph representing a software system. However, the graph representation of software captures only one-dimensional relations between software components. To get a finer description of the software structure higher-dimensional relations should be considered. Here, we will provide details on how topological space in terms of its higher-dimensional relations in complex software systems structures can be applied.

Hence, one step further into the structure of the software system and its components is to find groups of components tied together. Such groups may be viewed as simplices, just like two connected software components were viewed as edges (1-dimensional simplices). This gives the simplicial complex of a software system. It can be viewed as a higher-dimensional version of a graph. The motifs, discussed in the previous sections can also be viewed as such higher-dimensional objects in the software structure.

The simplicial complex of a software system is a discretization of some topological space. The topological invariants of that space may be viewed as topological invariants of the software. These invariants can be computed directly from the simplicial complex of software. They may be viewed as approximations of the software structure. The key point of this approach to software system structure is that one should approximate the structure by topological invariants obtained from the simplicial complex of the software system. The topological approach ignores the scale of the system and concentrates only on its shape. The topological analysis can detect hidden similarities and differences in software system structures.

In the analogy to the zoology and paleontology elaborated above, the topological study of software is entirely within the paleontology side. The software is like a dinosaur skeleton waiting for us to explore and extrapolate and try to figure out the properties of the whole animal. We will explain the first steps in that endeavour in the following sections.

4 Topological Algorithms

This section is devoted to the description of the topological algorithm which will be applied to the software structure. It is very well known algorithm and can be viewed as the "hello world" of topological analysis of software. The reference for this material could be any textbook devoted to computational topology, such as [10].

The algorithm presented here is the most basic algorithm of computational topology, and meant as an invitation to dive into the field of topological data analysis. It is the basic step in the study of computational algebraic topology, in particular, the homology, cohomology, persistant homology, among other.

The section should be read in combination with Sect. 5, because the latter contains the running example. The notions and operations described here are performed and explained in detail on the running example in Sect. 5.

The main concepts introduced here are the basic notions of homological algebra. The ultimate goal is to introduce the Betti numbers, i.e., the ranks of homology groups, associated to a simplicial complex. For simplicity of exposition, the underlying field is the field $\mathbb{Z}_2 = \{0, 1\}$ of two elements with addition and multiplication modulo two. The notions required and defined in this section are summarized in Table 1 for convenience of the reader.

The importance of homology groups, and the associated Betti numbers, lies in the fact that they encode certain topological information about the space, or in our case the software graph. These are mostly related to connectedness properties of the topological space whose "skeleton" is the software graph in question. Although the homology groups over $\mathbb{Z}_2$ considered in these lectures capture only limited topological information, they provide a nice introduction to

Notion	Symbol	Definition
Simplicial complex	Calligraphic letters $\mathcal{K}$	A family of non-empty sets which contains all non-empty subsets of its members
Simplex	σ, τ,...	Members of a simplicial complex
p-simplex		Simplex of dimension p, i.e., containing $p+1$ elements
p-chain	c, d,...	Formal sums of simplices of the same dimension p
Boundary operator	∂	The linear operator defined on chains as the chain of lower dimension given as the formal sum of the boundary faces
p-boundary operator	∂_p	The boundary operator acting on p-chains
Cycles	z, x, y,...	Chains with zero boundary
Boundaries	b, a,...	Chains obtained as boundaries of chains of higher dimension
Rank of p-cycles	z_p	The basis two logarithm of the number of p-cycles
Rank of p-boundaries	b_p	The basis two logarithm of the number of p-boundaries
Betti number	β_p	The non-negative integer obtained as $\beta_p = z_p - b_p$

Table 1: Summary of basic notions

the subject of topological data analysis. In particular, they are computationally accessible and the students can make their one code for computing the Betti numbers.

4.1 Simplicial complex

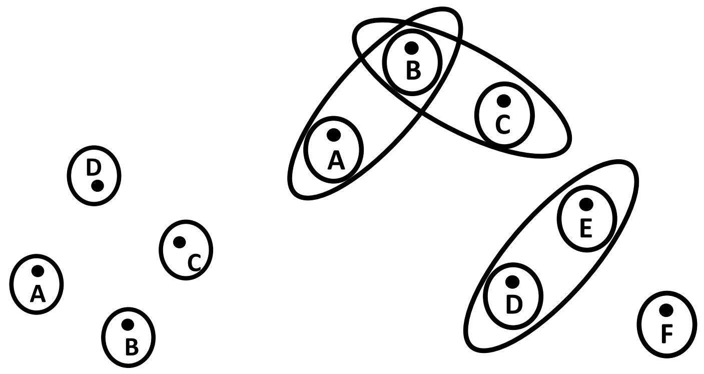
The definition of a simplicial complex is essentially very simple. The simplicial complex is just a bunch of sets, but whenever some set is in the bunch, then all its non-empty subsets are in the same bunch. It is as simple as that.

Formally, a finite family $\mathcal{K}$ of non-empty finite sets is called a simplicial complex if every non-empty subset τ of any set σ in the family $\mathcal{K}$ is also a member of the family $\mathcal{K}$ (called a face of σ), i.e.,

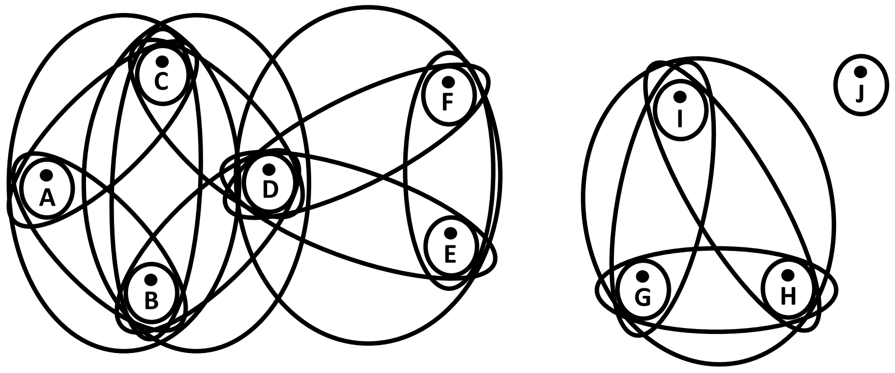
$$\text{if } \sigma \in \mathcal{K} \text{ and } \emptyset \neq \tau \subseteq \sigma, \text{ then } \tau \in \mathcal{K}.$$

Note that here Greek letters denote sets. This is usual in the study of simplicial complices.

The sets in a simplicial complex are called simplices. The elements of sets in a simplicial complex are usually called points. In other words, every simplex consists of points. The number of points in a simplex determines its dimension, sometimes also called the degree of a simplex. More precisely, a simplex with



(a) Simplicial complex $\mathcal{K}_a$ (b) Simplicial complex $\mathcal{K}_b$



(c) Simplicial complex $\mathcal{K}_c$

Fig. 9: Examples of simplicial complices viewed as families of sets.

$p+1$ points is of dimension p and usually referred to as a p-simplex. Thus, a p-simplex is a set in $\mathcal{K}$ containing exactly $p+1$ points, so that

0-simplices are just the points,
1-simplices contain two points (sometimes called segments),
2-simplices contain three points (sometimes called triangles),
...
p-simplices contain $p+1$ points.

Examples of simplicial complices, viewed as families of sets, are given in Fig. 9.

The first simplicial complex $\mathcal{K}_a$, given in Fig. 9a, consists of four sets with one element in each of these sets, that is, $\mathcal{K}_a$ is the family of sets

$$\mathcal{K}_a = \Big\{\{A\}, \{B\}, \{C\}, \{D\}\Big\}.$$

Thus, it contains four 0-simplices (points), and no higher-dimensional simplices.

The second simplicial complex $\mathcal{K}_b$, given in Fig. 9b, contains 1-simplices. As shown in the figure, it is the family of sets

$$\begin{aligned}\mathcal{K}_b = \Big\{&\{A,B\},\{B,C\},\{D,E\},\\ &\{A\},\{B\},\{C\},\{D\},\{E\},\{F\}\Big\},\end{aligned}$$

where the 1-simplices (segments) are listed in the first row and the 0-simplices (points) in the second row.

The third simplicial complex $\mathcal{K}_c$, given in Fig. 9c, is more complicated. It contains the 2-simplices. As shown in the figure, $\mathcal{K}_c$ is the family of sets

$$\begin{aligned}\mathcal{K}_c = \Big\{&\{A,B,C\},\{B,C,D\},\{D,E,F\},\{G,H,I\},\\ &\{A,B\},\{A,C\},\{B,C\},\{B,C\},\{C,D\},\{D,E\},\\ &\{D,F\},\{E,F\},\{G,H\},\{G,I\},\{H,I\},\\ &\{A\},\{B\},\{C\},\{D\},\{E\},\{F\},\{G\},\{H\},\{I\},\{J\}\Big\}\end{aligned}$$

where is the 2-simplices are listed in the first row, the 1-simplices in the second and third row, and the 0-simplices in the last row. Observe that all non-empty subsets of every set in the family are also members of the family. For example, since the 2-simplex $\{A,B,C\}$ is in the simplicial complex $\mathcal{K}_c$, all its non-empty subsets $\{A,B\}$, $\{A,C\}$, $\{B,C\}$, $\{A\}$, $\{B\}$, $\{C\}$ are also simplices in $\mathcal{K}_c$.

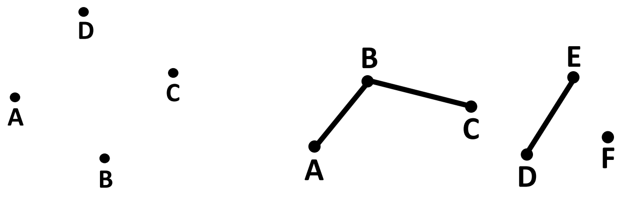
These figures of simplicial complices become very difficult to follow if the simplical complex contains higher-dimensional simplices. Already Fig. 9c is quite messy. Therefore, we will develop data structures for handling the simplicial complices. These are introduced in Sect. 5. The simplicial complex and its simplices in the running example of Sect. 5 are constructed inductively in Sect. 5.4, starting with the lowest dimensions zero and one in Sect. 5.3.

4.2 Geometric viewpoint

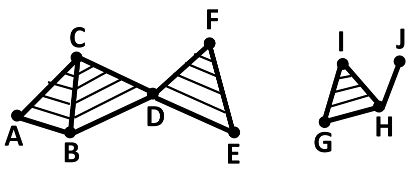
Simplicial complices arise from geometry (or more precisely topology), and may be interpreted as the family of polyhedra in a sufficiently high-dimensional space. That is where the terminology using points, faces, boundaries etc, comes from. The geometric viewpoint is a way of visualizing the set-based definition of simplicial complex, introduced in Sect. 4.1, but not a suitable form of representation for computational purposes.

The simplicial complices of Fig. 9 are represented from the geometric viewpoint in Fig. 10. The sets of two points are represented by real geometric segments, the sets of three points by triangles, and those of four points by tetrahedra. The problem of representing the higher-dimensional simplices remains, as it requires more than three dimensions.

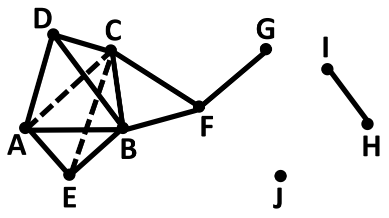
Observe how the simplicial complex $\mathcal{K}_a$, from the geometric point of view, is represented in Fig. 10a by four points A, B, C, D in space. The simplicial



(a) Simplicial complex $\mathcal{K}_a$ (b) Simplicial complex $\mathcal{K}_b$



(c) Simplicial complex $\mathcal{K}_c$



(d) Simplicial complex $\mathcal{K}_d$

Fig. 10: Examples of simplicial complices from the geometric viewpoint.

complex $\mathcal{K}_b$ is presented in Fig. 10b. Its 1-simplices are represented by line segments $\overline{AB}$, $\overline{BC}$, $\overline{DE}$, while 0-simplices are points. The simplicial complex $\mathcal{K}_c$ is presented in Fig. 10c. Its 2-simplices are represented by the triangles $\triangle ABC$, $\triangle BCD$, $\triangle DEF$, $\triangle GHI$, its 1-simplices are the segments in the figure, and the 0-simplices the points. The figures of simplicial complices from the geometric point of view seem simpler than the figures of the same simplicial complices as families of sets above. However, the limitation of representing the higher-dimensional simplices remain.

The last simplicial complex $\mathcal{K}_d$, given in Fig. 10d, contains 3-simplices. These are the tetrahedra $ABCD$ and $ABCE$. The 2-simplices are triangles $\triangle ABC$, $\triangle ABD$, $\triangle ABE$, $\triangle ACD$, $\triangle ACE$, $\triangle BCD$, $\triangle BCE$. Observe that the triangle $\triangle BCF$ is not a 2-simplex as it is not shaded, although all of its sides are 1-simplices. The 1-simplices are all segments in the figure, and 0-simplices are all points in the figure.

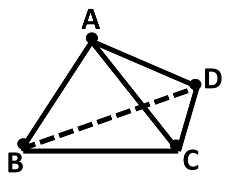


Fig. 11: The tetrahedron $ABCD$ representing a simplicial complex.

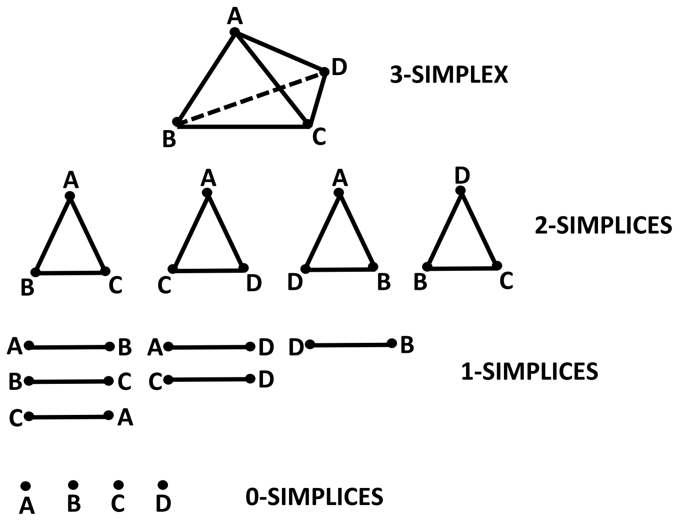


Fig. 12: The simplices of the simplicial complex represented by the tetrahedron $ABCD$.

As another example, consider a tetrahedron $ABCD$ given in Fig. 11. It represents a 3-simplex. In the abstract approach taken above, this 3-simplex is just the set

$$\sigma = \{A, B, C, D\}$$

of vertices of the tetrahedron. Notice that the geometrical dimension of a tetrahedron coincides with the simplicial dimension. That is the reason why simplices with $p + 1$ points are considered to be p-dimensional.

The faces of the tetrahedron are the four triangles $\triangle ABC$, $\triangle ABD$, $\triangle BCD$ and $\triangle CAD$, as shown in Fig. 12. These are the 2-simplices that are subsimplices of the tertrahedron. In the abstract setting, these are the sets

$$\{A, B, C\}, \{A, B, D\}, \{B, C, D\}, \{C, A, D\},$$

of the vertices of the triangles. Observe that these are all subsets with three elements of the 3-simplex $\sigma = \{A, B, C, D\}$. In geometric language, these faces form the boundary of the tetrahedron. The notion of boundary will play a prominent role in the abstract simplicial complices as well.

Going one step further, the six edges $\overline{AB}$, $\overline{BC}$, $\overline{CA}$, $\overline{AD}$, $\overline{BD}$, $\overline{CD}$ of the tetrahedron, shown in Fig. 12, represent 1-simplices. These edges are at the same time the sides of the triangle faces of the tetrahedron. In the abstract setting, the 1-simplices are just the sets of endpoint of these edges, i.e.,

$$\{A, B\}, \{B, C\}, \{C, A\}, \{A, D\}, \{B, D\}, \{C, D\}.$$

Observe again that these are all subsets with two elements of the 3-simplex $\sigma = \{A, B, C, D\}$, but they are also subsets of some of the 2-simplices arising from triangle faces of the tetrahedron.

Finally, the four vertices A, B, C, D of the tetrahedron represent 0-simplices. They are endpoints of the edges and also vertices of the triangle faces of the tetrahedron. In the abstract setting, they form the sets

$$\{A\}, \{B\}, \{C\}, \{D\}.$$

These are again all subsets with one element of the 3-simplex $\sigma = \{A, B, C, D\}$, but they are also subsets of some of the 2-simplices represented by edges and 3-simplices represented by triangle faces of the tetrahedron.

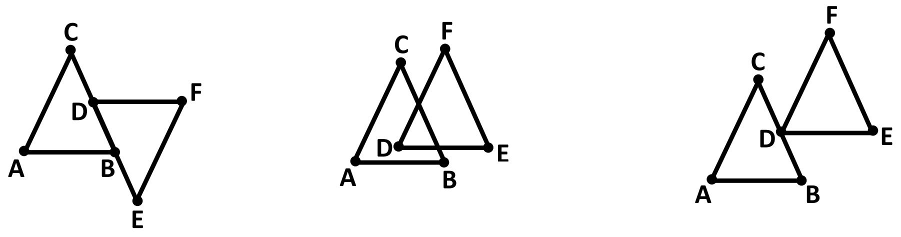


Fig. 13: Non-examples of simplicial complices.

Although one may think of a simplex as a polyhedron, and its subsimplices as its faces of all dimensions (vertices, edges, faces, higher-dimensional faces), it is easier, and computationally more convenient, to work with an abstract data structure such as our definition of the simplicial complex. In that way, we avoid geometric concerns. For example, there could be several intersecting polyhedra in a simplicial complex, and they must intersect only in their faces. More precisely, the intersection of any two such polyhedra must be a face of both. But in geometry, polyhedra may intersect in different geometric objects. See, for example, the intersections of two triangles in Fig. 13. Only if the polyhedra intersect in a face or an edge or a vertex, they represent a simplicial complex. This is easier to handle by considering a bunch of sets, than looking at the geometric picture and properties of polyhedra.

4.3 Simplicial complex of a (software) graph

There are several ways to assign a simplicial complex to a graph. We explain here the simplest way as an example, although this is too naive for serious applications. Our goal here is to explain a topological algorithm, so this simple approach will do. Be aware that our approach could result in computationally highly demanding algorithm in the case of an arbitrary graph. However, the nature of software graphs is such that the computation will be feasible. As already mentioned in Sect. 4.1, the construction is illustrated in the running example of Sect. 5, more precisely, in Sect. 5.3 and Sect. 5.4.

Let $G = (V, E)$ be a graph with the set of vertices V and the set of edges E. We consider only simple graphs, i.e., G is undirected, unweighted, has no loops and no multiple edges. Recall that a loop is an edge with equal endpoints, and a multiple edge refers to the existence of several edges with the same endpoints. An example is given in Fig. 17 and explained in Sect. 5.

A p-simplex in the graph G is defined as any subgraph containing $p + 1$ vertices such that each pair of vertices is connected by an edge. Recall that such subgraph is called a complete graph with $p + 1$ vertices. In applications, such subgraphs are often referred to as cliques in a graph.

More precisely, in our approach, the simplices of the simplicial complex associated to a given graph are the following:

0-simplices are just vertices (complete subgraphs with 1 vertex),
1-simplices are just edges (complete subgraphs with 2 vertices),
2-simplices are triangles (complete subgraphs with 3 vertices),
3-simplices are tetrahedra (complete subgraphs with 4 vertices),
...
p-simplices are complete subgraphs with $p + 1$ vertices.

Observe that all subgraphs of a complete graph are also complete. Therefore, we really obtained a simplicial complex, because the faces of a simplex are indeed simplices.

4.4 Chains

We proceed with an arbitrary simplicial complex $\mathcal{K}$. In applications, it will be the simplicial complex assigned to a software graph. Examples of all the notions introduced here are given in the running example of Sect. 5. However, since the notions in the running example are all expressed in an appropriate basis given by simplices, the chains do not appear explicitly in Sect. 5. They are hidden in the linear algebra formalism.

The motivation for introducing chains and their addition is to have a linear algebra formalism which allows the consideration of several simplices of the same dimension as a single object. This will allow the study of higher-dimensional topological structures, and in particular the Betti numbers, using linear algebra.

Let S_p denote the family of all p-simplices in the simplicial complex $\mathcal{K}$. Denote by n_p the number of p-simplices, i.e., $n_p = |S_p|$ is the cardinality of S_p. Write

$$S_p = \{\sigma_1, \sigma_2, \ldots, \sigma_{n_p}\}$$

for the p-simplices in $\mathcal{K}$.

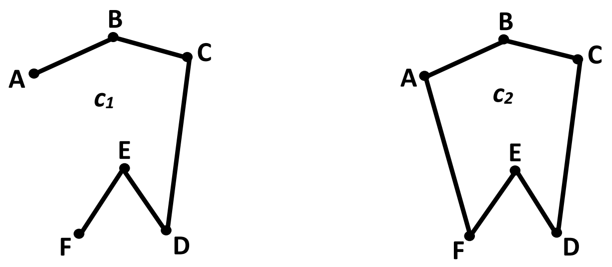


Fig. 14: Examples of 1-chains c_1 and c_2.

A p-chain in $\mathcal{K}$ is any subfamily of S_p. In other words, any choice of any number of p-simplices produces a p-chain. Even the empty choice, in which none of the p-simplices is chosen, is allowed.

Examples of 1-chains are given in Fig. 14. The 1-chain c_1 on the left-hand side in the figure consists of 1-simplices $\overline{AB}$, $\overline{BC}$, $\overline{CD}$, $\overline{DE}$ and $\overline{EF}$, while the 1-chain c_2 on the right-hand side in the figure consists of the same 1-simplices with $\overline{FA}$ as an extra 1-simplex.

There is a convenient way to write p-chains using formal sums of p-simplices with coefficients in $\mathbb{Z}_2 = \{0, 1\}$. The coefficients encode our choice of p-simplices in a p-chain. If the coefficient of a p-simplex is zero, then it is not chosen in a p-chain, while if the coefficient is one, then it is chosen to be in a p-chain.

More precisely, a p-chain c in $\mathcal{K}$ can be written as a formal sum

$$c = \sum_{i=1}^{n_p} a_i \sigma_i,$$

where $a_i \in \mathbb{Z}_2 = \{0, 1\}$. If the coefficient $a_i = 0$, then σ_i is not in the p-chain c, while if $a_i = 1$, then σ_i is in the p-chain c. The 1-chains in Fig. 14 can be written as

$$c_1 = \overline{AB} + \overline{BC} + \overline{CD} + \overline{DE} + \overline{EF}$$
$$c_2 = \overline{AB} + \overline{BC} + \overline{CD} + \overline{DE} + \overline{EF} + \overline{FA}$$

in the formal sum notation.

Another point of view, perhaps more familiar, is to interpret a p-chain c in $\mathcal{K}$ as a sequence of coefficients

$$c = (a_1, a_2, \ldots, a_{n_p}),$$

where $a_i \in \mathbb{Z}_2 = \{0, 1\}$, so that a p-chain is any n_p-bit word. The coefficient a_i again indicates whether the p-simplex σ_i is in the chain c or not.

Let C_p denote the set of all p-chains in the simplicial complex $\mathcal{K}$. The addition of p-chains is defined, either as the mod 2 addition of formal sums, or as the exclusive or operation on n_p-bit words. Let

$$c = \sum_{i=1}^{n_p} a_i \sigma_i = (a_1, a_2, \ldots, a_{n_p}),$$

$$c' = \sum_{i=1}^{n_p} b_i \sigma_i = (b_1, b_2, \ldots, b_{n_p})$$

be two p-chains in C_p. Then, their sum is given as

$$\begin{aligned} c + c' &= \sum_{i=1}^{n_p} (a_i +_2 b_i)\sigma_i \\ &= (a_1 \oplus b_1, a_2 \oplus b_2, \ldots, a_{n_p} \oplus b_{n_p}), \end{aligned}$$

where $+_2$ stands for the mod 2 addition and $\oplus$ stands for the exclusive or operation. The sum of the two 1-chains in Fig. 14 is

$$c_1 + c_2 = \overline{FA},$$

because the other 1-simplices appear in both 1-chains, so that they cancel in mod 2 addition.[2]

Observe that the cardinality $|C_p| = 2^{n_p}$, because it is the number of n_p-bit words. The exponent n_p is called the rank of C_p. The reason for exclusive-or, i.e., mod 2, addition of chains is used here, because we restrict the scope of this lecture notes to the simplest case of the field $\mathbb{Z}_2$ of two elements. The analogous theory can be developed over other fields and rings, such as the ring of integers $\mathbb{Z}$. However, in that case the boundary operator, introduced in Sect. 4.5 below, must be defined in a different way, using certain alternating sums of boundary faces. This is necessary in order to satisfy the fundamental property of boundary operators, explained in Sect. 4.7.

4.5 Boundary operator

The boundary operator is the glue that fits together all the simplices of different dimensions in a simplicial complex. The p-boundary operator ∂_p assigns to each p-simplex its boundary, i.e., the family of its $(p-1)$-simplex faces. But such

[2] We mention, for the record, that the set C_p of all p-chains in the simplicial complex $\mathcal{K}$ forms an Abelian group with addition. However, this fact and the algebraic notion of groups will not be necessary for understanding the rest of the paper.

family is a $(p-1)$-chain. Given a p-simplex $\sigma_i \in S_p$, the action of the p-boundary operator can be expressed as

$$\partial_p \sigma_i = \sum_{\substack{\tau \in S_{p-1} \\ \tau \subseteq \sigma_i}} \tau \in C_{p-1},$$

which is a sum of $(p-1)$-simplices τ, and thus a $(p-1)$-chain in C_{p-1}. In the running example of Sect. 5, the boundary operators are determined in their matrix representation as part of the construction of the simpicial complex in Sect. 5.3 and Sect. 5.4.

For example, the 3-boundary ∂_3 of the 3-simplex given by the tetrahedron $ABCD$ in Fig. 11 is given as

$$\partial_3(ABCD) = \triangle ABC + \triangle ACD + \triangle ADB + \triangle BCD,$$

which is the 2-chain given as the sum of 2-simplex faces in Fig. 12 of the tetrahedron $ABCD$.

More generally, given a p-chain

$$c = \sum_{i=1}^{n_p} a_i \sigma_i \in C_p,$$

the p-boundary operator ∂_p acts as

$$\begin{aligned}\partial_p c &= \partial_p \left(\sum_{i=1}^{n_p} a_i \sigma_i \right) \\ &= \sum_{i=1}^{n_p} a_i \partial_p \sigma_i \in C_{p-1},\end{aligned}$$

where

$$\partial_p \sigma_i = \sum_{\substack{\tau \in S_{p-1} \\ \tau \subseteq \sigma_i}} \tau \in C_{p-1},$$

as above. Hence, ∂_p is a map from C_p to C_{p-1}. It is now clear from the definition how the boundary operator glues together information on simplices and chains of different dimensions.

For example, the 1-boundary operator ∂_1 applied to the 1-chains c_1 and c_2 in Fig. 14 is computed as follows

$$\begin{aligned}\partial_1 c_1 &= \partial_1 \left(\overline{AB} + \overline{BC} + \overline{CD} + \overline{DE} + \overline{EF} \right) \\ &= \partial_1 \overline{AB} + \partial_1 \overline{BC} + \partial_1 \overline{CD} + \partial_1 \overline{DE} + \partial_1 \overline{EF} \\ &= (A + B) + (B + C) + (C + D) + (D + E) + (E + F) \\ &= A + F\end{aligned}$$

which is the 0-chain given as the sum of 0-simplices A and F, while

$$\begin{aligned}\partial_1 c_2 &= \partial_1 \left(\overline{AB} + \overline{BC} + \overline{CD} + \overline{DE} + \overline{EF} + \overline{FA}\right)\\ &= \partial_1\overline{AB} + \partial_1\overline{BC} + \partial_1\overline{CD} + \partial_1\overline{DE} + \partial_1\overline{EF} + \partial_1\overline{FA}\\ &= (A+B)+(B+C)+(C+D)+(D+E)+(E+F)+(F+A)\\ &= 0\end{aligned}$$

which is the empty 0-simplex denoted by zero. The boundary operator computes the boundary of the simplices in a chain expressed as a sum over $\mathbb{Z}_2$, i.e., with mod 2 addition. The boundary operator gives 0 if the simplices form a cycle, as in the example of c_2.

At this point, it becomes clear why the formal sum notation for p-chains is much more convenient than the n_p bit words approach. The reason is that, in the formal sum notation, it is not necessary to maintain the order of p-simplices. Given a p-chain c in C_p, one may write

$$c = \sum_{\sigma \in S_p} a_\sigma \sigma,$$

where $a_\sigma \in \mathbb{Z}_2 = \{0,1\}$ is the coefficient of the p-simplex σ. As above $a_\sigma = 1$ if σ belongs to c, and $a_\sigma = 0$ if σ does not belong to c. The order of σ is irrelevant.

This advantage of the formal sum notation is useful in the definition of the p-boundary operator. It is not necessary to specify which simplices τ are in the boundary when defining and using the p-boundary operator. However, as we will see below, for explicit calculation of topological invariants, it is convenient to have the order of p-simplices fixed.

4.6 Matrix of the boundary operator

The convenient way to view the boundary operator is as the linear operator on the vector spaces of chains. In this context, the matrix of the boundary operators as a linear operator can be introduced. That is the subject of this section. In the running example of Sect. 5, the matrix of the boundary operators are determined inductively in Sect. 5.4, starting with the lowest dimension in Sect. 5.3.

The p-chains in C_p may be viewed as linear combinations with coefficients in $\mathbb{Z}_2 = \{0,1\}$ of simplices in S_p. Hence, they form a vector space over the field $\mathbb{Z}_2$ of two elements. The basis of C_p as a vector space over $\mathbb{Z}_2$ is the set S_p of all p-simplices. Thus, the dimension of C_p over $\mathbb{Z}_2$ is the number $n_p = |S_p|$ of p-simplices.

By the very definition of the p-boundary operator, it is a linear operator from C_p to C_{p-1} as vector spaces over $\mathbb{Z}_p$, because it respects the linear combinations

$$\partial_p \left(\sum_{i=1}^{n_p} a_i \sigma_i \right) = \sum_{i=1}^{n_p} a_i \partial_p \sigma_i,$$

for any $a_i \in \mathbb{Z}_2$.

As any linear operator, the p-boundary operator ∂_p can be represented by a matrix. The matrix of the p-boundary operator ∂_p has n_{p-1} rows, because it is the dimension of C_{p-1}, and n_p columns, because it is the dimension of C_p. It contains only 0's and 1's, because the vector spaces are over $\mathbb{Z}_2 = \{0, 1\}$. The rows represent $(p-1)$-simplices in S_{p-1}, and the columns represent p-simplices in S_p, in a fixed order.

The j-th column represents the p-simplex $\sigma_j \in S_p$. It contains 1 in the rows representing its $(p-1)$-simplex faces in S_{p-1}, and 0 in the rows which represent $(p-1)$-simplices in S_{p-1} which are not its faces. Every p-simplex has exactly $p+1$ faces, so that each column contains precisely $p+1$ ones. More precisely, the element at the crossing of the i-th row and the j-th column equals 1 if the i-th simplex in S_{p-1} is a $(p-1)$-simplex face of the j-th simplex in S_p, and it equals 0 otherwise. More precisely,

$$\partial_p = \begin{bmatrix} d_{1,1} & \dots & d_{1,n_p} \\ & \ddots & \\ \vdots & d_{i,j} & \vdots \\ & & \ddots \\ d_{n_{p-1},1} & \dots & d_{n_{p-1},n_p} \end{bmatrix},$$

where

$$d_{i,j} = \begin{cases} 1, \text{ if the } i\text{-th } (p-1)\text{-simplex } \tau_i \in S_{p-1} \text{ is a face} \\ \quad \text{of the } j\text{-th } p\text{-simplex } \sigma_j \in S_p, \\ 0, \text{ otherwise.} \end{cases}$$

4.7 Boundary operators as differentials

In this section, we explain the most fundamental property of boundary operators. It is the fact that the boundary of a boundary is zero.

The boundary operators in different dimensions fit into this picture

$$\dots \xrightarrow{\partial_{p+2}} C_{p+1} \xrightarrow{\partial_{p+1}} C_p \xrightarrow{\partial_p} C_{p-1} \xrightarrow{\partial_{p-1}} \dots \xrightarrow{\partial_2} C_1 \xrightarrow{\partial_1} C_0.$$

The fundamental property of the boundary operators is that the composition of consecutive boundary operators is the zero operator, i.e.,

$$\partial_p \circ \partial_{p+1} = 0$$

for all p. In other words, the boundary of the boundary of any chain is 0.

This fundamental property of the boundary operators is the defining property of differentials of a simplicial complex. Hence, by the definition of differentials, this property of the boundary operators means that they are differentials of the simplicial complex.

The proof of the fundamental property of boundary operators follows from the fact that every $(p-1)$-simplex face of a $(p+1)$-simplex is at the same time a face of exactly two of its p-simplex faces, and since $2 = 0 \bmod 2$, the resulting chain is the zero chain. This can be easily checked in the abstract view of the simplicial complex. Given a $(p+1)$-simplex as a set of $p+1$ points, its $(p-1)$-simplex face is a subset of $p-1$ points. In other words, the $(p-1)$-simplex face is obtained by removing two points from the $(p+1)$-simplex. But then, adding each of the removed points to the $(p-1)$-simplex face produces exactly two p-simplex faces of the $(p+1)$-simplex which share the $(p-1)$-simplex face. Hence, the number of appearances of any $(p-1)$-simplex face in the boundary of the boundary of the $(p+1)$-simplex is even, which equals 0 in mod 2 arithmetic of $\mathbb{Z}_2$.

4.8 Cycles and boundaries

The cycles and boundaries in a simplicial complex are special types of chains. They are defined in terms of the boundary operators. In the running example of Sect. 5, the cycles and boundaries are not determined explicitly. Instead, only their numbers are determined in Sect. 5.5 from the matrices of boundary operators using the linear algebra notions of rank and defect of a linear operator.

A p-cycle z is a p-chain with zero boundary, i.e.,

$$\partial_p z = 0.$$

The set of all p-cycles is denoted Z_p. It is a subset of the set C_p of all p-chains.[3]

An example of 1-cycle is the 1-chain c_2 in Fig. 14, because we already calculated in Sect. 4.5 its 1-boundary and obtained $\partial_1 c_2 = 0$ is the empty 0-chain. The other 1-chain c_1 in Fig. 14 is not an 1-cycle, because its 1-boundary $\partial_1 c_1 = A + F$ is not the empty 0-chain. The figure also explains where the name of the cycles comes from. The 1-cycle c_2 is represented by a closed loop, while the 1-chain c_1, which is not an 1-cycle, is not.

The number $|Z_p|$ of p-chains is a power of two

$$|Z_p| = 2^{z_p},$$

where the exponent z_p is called the rank of Z_p. In the special case of $p = 0$, there is no boundary operator ∂_0, but we make the convention that all 0-chains are 0-cycles, i.e., $Z_0 = C_0$, and the rank of Z_0 is $z_0 = n_0$.

A p-boundary b is a p-chain which is a boundary of some $(p+1)$-chain, i.e.,

$$b = \partial_{p+1} c$$

for some $c \in C_{p+1}$. The set of all p-boundaries is denoted B_p. It is a subset of the set C_p of all p-chains.[4]

[3] For the record, the set Z_p of p-cycles forms an Abelian group under addition of chains.

[4] For the record, the set B_p of p-boundaries forms an Abelian group under addition of chains.

The number $|B_p|$ of p-boundaries is also a power of two

$$|B_p| = 2^{b_p},$$

where the exponent b_p is called the rank of B_p.

If t is the top dimension of a simplex in a simplicial complex, i.e., there are no p-simplices of dimension $p > t$, then there are no t-boundaries, because there is no $(t+1)$-simplex in the simplicial complex. Hence, B_t is trivial and the rank $b_t = 0$ for the top dimension t.

The fundamental property of the boundary operators implies that every p-boundary is a p-cycle, because the boundary of the boundary is always zero. More precisely, if $b = \partial_{p+1}c$ is any p-boundary, then

$$\partial_p b = \partial_p(\partial_{p+1}c) = (\partial_p \circ \partial_{p+1})c = 0,$$

which means that b is a p-cycle. Thus,

$$B_p \subseteq Z_p$$

for all p.

4.9 Homology

The p-th homology group H_p of a simplicial complex counts the p-cycles which are not obtained as p-boundaries, i.e., the "true" cycles among p-chains.[5] For our purposes, it is sufficient to know that the number $|H_p|$ of elements in the p-th homology group is also a power of two, and that its rank is the difference between the ranks of of Z_p and B_p. In the running example of Sect. 5, the Betti numbers are determined in the final step of Sect. 5.6. If we write

$$|H_p| = 2^{\beta_p},$$

then the rank β_p of H_p is called the p-th Betti number. It is obtained by the formula

$$\begin{aligned}
\beta_0 &= z_0 - b_0 = n_0 - b_0, & \\
\beta_p &= z_p - b_p, & 1 \le p \le t-1, \\
\beta_t &= z_t - b_t = z_t, & \\
\beta_q &= 0, & q > t,
\end{aligned}$$

where t is the top dimension of a simplex in the simplicial complex. All these values can be read off from the matrix of ∂_p, as we shall see in the examples below in Sect. 5. However, in order to provide intuition for the chains representing elements of the homology groups over $\mathbb{Z}_2$, we begin here with a simple illustrative example.

[5] Formally, the p-th homology group is defined as the quotient group $H_p = Z_p/B_p$, which is again an Abelian group.

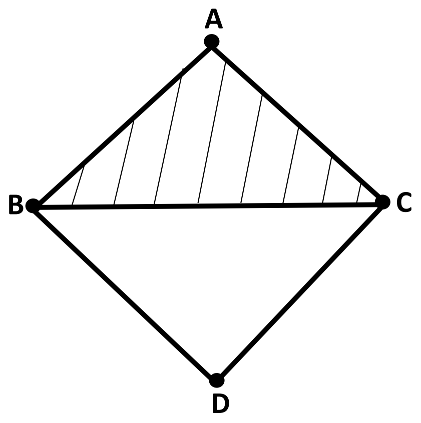


Fig. 15: Simplicial complex $\mathcal{K}$ in the example of Sect. 4.9

Let $\mathcal{K}$ be the simplicial complex given in Fig. 15. The shaded triangle in the figure is a 2-simplex in $\mathcal{K}$, while the unshaded one is not in $\mathcal{K}$. Hence, the simplicial complex $\mathcal{K}$ consists of the following simplices

$$\begin{aligned}
S_2 &= \{\triangle ABC\},\\
S_1 &= \{\overline{AB}, \overline{AC}, \overline{BC}, \overline{BD}, \overline{CD}\},\\
S_0 &= \{A, B, C, D\}.
\end{aligned}$$

The chain representing an element in homology group is best exhibited in dimension one. Hence, we first determine 1-cycles.

The 1-boundary operator acts on an arbitrary 1-chain as

$$\begin{aligned}
&\partial_1(x\overline{AB} + y\overline{AC} + z\overline{BC} + u\overline{BD} + v\overline{CD})\\
&\qquad = x(A+B) + y(A+C) + z(B+C) + u(B+D) + v(C+D)\\
&\qquad = (x+y)A + (x+z+u)B + (y+z+v)C + (u+v)D,
\end{aligned}$$

where $x, y, z, u, v \in \mathbb{Z}_2$ are arbitrary coefficients. The condition for 1-cycles is that the obtained 0-chain is trivial, i.e., all the coefficients must be zero. Solving the system of equations over $\mathbb{Z}_2$ gives

$$\begin{aligned}
y &= x\\
v &= u\\
z &= x + u,
\end{aligned}$$

where $x, u \in \mathbb{Z}_2$ are arbitrary. Hence, there are four 1-cycles, and the rank of Z_1 is $z_1 = 2$. More precisely, the 1-cycles are all 1-chains of the form

$$x\overline{AB} + x\overline{AC} + (x+u)\overline{BC} + u\overline{BD} + u\overline{CD},$$

which can be rearranged as

$$x(\overline{AB} + \overline{AC} + \overline{BC}) + u(\overline{BC} + \overline{BD} + \overline{CD}),$$

where $x, u \in \mathbb{Z}_2$ are arbitrary.

The next task is to determine the 1-boundaries. The only 2-simplex is the triangle $\triangle ABC$, hence the 2-boundary operator acts on an arbitrary 2-chain as

$$\partial_2(k\triangle ABC) = k(\overline{AB} + \overline{AC} + \overline{BC}),$$

so that the only non-trivial 1-boundary is the 1-chain

$$\overline{AB} + \overline{AC} + \overline{BC},$$

and the rank of B_1 is $b_1 = 1$.

From the above, we can easily compute the Betti number in dimension one, which is

$$\beta_1 = z_1 - b_1 = 1.$$

However, in this example we would like to provide an exemplary 1-chain representing the non-trivial element of the homology group H_1. Comparing the obtained 1-boundary with the description of 1-cycles, we see that 1-cycle in the first bracket, the one with coefficient x, is actually an 1-boundary, and thus not a non-trivial element of H_1. This implies that the non-trivial element in H_1 is represented by the other bracket in the description of 1-cycles above. It is the 1-cycle

$$\overline{BC} + \overline{BD} + \overline{CD}.$$

In this way we determined an example of a cycle, which is not a boundary.

There is another subtlety arising from the quotient group definition of H_1. Namely, there could be more than one 1-cycle representing the same element in H_1. In our example, this happens with the 1-cycle

$$(\overline{AB} + \overline{AC} + \overline{BC}) + (\overline{BC} + \overline{BD} + \overline{CD}) = \overline{AB} + \overline{AC} + \overline{BD} + \overline{CD},$$

which is the mod 2 sum of the two 1-cycles considered above. This 1-cycle is not an 1-boundary by itself, so that it should also represent a non-trivial element of H_1. However, the mod 2 difference between the two representatives is precisely the 1-boundary $\overline{AB} + \overline{AC} + \overline{BC}$, so that, up to boundary, the two 1-cycles are equal, and therefore represent the same element in H_1.

4.10 Example 1: Homology of a tetrahedron

We consider now a simple geometric example, which can be done directly by hand and can be observed in pictures. It is the example of a tetrahedron, which is already a 3-simplex, so that the simplicial complex of a tetrahedron consists of a single 3-simplex with all its subsimplices. This example is already described in Sect. 4.2 and depicted in Fig. 11 and Fig. 12. The sets S_p of p-simplices are already listed there, and their geometric presentation is discussed. For determining the homology of a tetrahedron and its Betti numbers, we only require the description of the boundary operators.

The 3-boundary operator ∂_3 acts on the set C_3 of 3-chains. Since there is only one 3-simplex $\{A, B, C, D\}$, i.e., the whole tetrahedron $ABCD$, all the 3-chains are of the form

$$c_3 = k \cdot ABCD,$$

where $ABCD = \{A, B, C, D\}$ denotes the tetrahedron as a 3-simplex and $k \in \mathbb{Z}_2$ is the coefficient. In other words, there are only two 3-chains: the empty chain and the chain $ABCD$. The action of the 3-boundary operator on the 3-chain c_3 is given by the formula

$$\partial_3 c_3 = k \cdot (\triangle BCD + \triangle ACD + \triangle ABD + \triangle ABC),$$

where the triangles in the brackets are 2-simplices that form the boundary of the tetrahedron ABCD.

From this formula we can read off the 3-cycles and 2-boundaries. There are no non-trivial 3-cycles, because $\partial_3 c_3 = 0$ only if the coefficient $k = 0$. Thus, c_3 is a 3-cycle if and only if it is trivial, i.e., the empty 3-chain. Since 3 is the top dimension of the simplicial complex of the tetrahedron, it follows that the homology group

$$H_3 = Z_3 = \{0\}$$

is trivial, and the Betti number $\beta_3 = 0$.

The 2-boundaries are the 2-chains obtained as the image of the 3-boundary operator. Hence, B_2 consists of

$$B_2 = \Big\{ k \cdot (\triangle BCD + \triangle ACD + \triangle ABD + \triangle ABC), \quad \text{where } k \in \mathbb{Z}_2 \Big\}.$$

In other words, there are only two 2-boundaries: the empty 2-chain and the 2-chain

$$\triangle BCD + \triangle ACD + \triangle ABD + \triangle ABC.$$

Thus, the rank of B_2 is $b_2 = 1$.

We should consider next the 2-boundary operator ∂_2, which acts on 2-chains. Since there are four 2-simplices, i.e., four triangles, in the simplicial complex, any 2-chain is of the form

$$c_2 = a \cdot \triangle BCD + b \cdot \triangle ACD + c \cdot \triangle ABD + d \cdot \triangle ABC,$$

where the triangles represent 2-simplices and $a, b, c, d \in \mathbb{Z}_2$ are the coefficients. Note that the number of 2-chains is $2^{n_2} = 2^4 = 16$.

The action of the 2-boundary operator on the 2-chain c_2 is given by the formula

$$\begin{aligned}\partial_2 c_2 =& a \cdot \big(\overline{BC} + \overline{CD} + \overline{DB}\big) + b \cdot \big(\overline{AC} + \overline{CD} + \overline{DA}\big) \\ &+ c \cdot \big(\overline{AB} + \overline{BD} + \overline{DA}\big) + d \cdot \big(\overline{AB} + \overline{BC} + \overline{CA}\big) \\ =& (c + d) \cdot \overline{AB} + (b + d) \cdot \overline{AC} + (b + c) \cdot \overline{AD} \\ &+ (a + d) \cdot \overline{BC} + (a + c) \cdot \overline{BD} + (a + b) \cdot \overline{CD}.\end{aligned}$$

Here in the first line we compute the boundary of each triangle separately, and then in the second line sum up according to segments. i.e., 1-simplices.

We can obtain from this formula the 2-cycles and 1-boundaries. The 2-cycles are those 2-chains c_2 for which $\partial_2 c_2 = 0$. In mod 2 arithmetic, this happens if and only if all the coefficients are zero, i.e.,

$$a = b = c = d = e = f.$$

Therefore, the set of 2-cycles is

$$Z_2 = \Big\{ a \cdot (\triangle BCD + \triangle ACD + \triangle ABD + \triangle ABC)\,, \quad \text{where } a \in \mathbb{Z}_2 \Big\}.$$

But this is equal to the set B_2 of 2-boundaries obtained above. Hence, the homology group

$$H_2 = \{0\}$$

is also trivial, and the Betti number $\beta_2 = 0$.

Finding all 1-boundaries by hand is a bit more involved. The trick is to determine first the 1-cycles, because every 1-boundary is at the same time an 1-cycle, by the fundamental property of differentials in a simplicial complex. Finding 1-boundaries among 1-cycles turns out to be easier.

Therefore, consider now the 1-boundary operator ∂_1 which acts on 1-chains. There are six 1-simplices in the simplicial complex, given by the edges of the tetrahedron. Thus, any 1-chain is of the form

$$c_1 = x \cdot \overline{AB} + y \cdot \overline{AC} + z \cdot \overline{AD} + u \cdot \overline{BC} + v \cdot \overline{BD} + w \cdot \overline{CD},$$

where the segments represent the 1-simplices and $x, y, z, u, v, w \in \mathbb{Z}_2$ are coefficients. The number of 1-chains is $2^{n_1} = 2^6 = 64$.

The action of the 1-boundary operator ∂_1 on c_1 is given by the formula

$$\begin{aligned} \partial_1 c_1 =& x \cdot (A + B) + y \cdot (A + C) + z \cdot (A + D) \\ & + u \cdot (B + C) + v \cdot (B + D) + w \cdot (C + D) \\ =& (x + y + z) \cdot A + (x + u + v) \cdot B + (y + u + w) \cdot C + (z + v + w) \cdot D, \end{aligned}$$

where in the first line we just determined the boundaries of the segments as the sum of their endpoints, and in the second line we summed up according to points, i.e., 0-simplices.

The 1-cycles are those 1-chains for which $\partial_1 c_1 = 0$, so that they are determined by the solutions of the system of linear equations

$$\begin{aligned} x + y + z &= 0 \\ x + u + v &= 0 \\ y + u + w &= 0 \\ z + v + w &= 0 \end{aligned}$$

in $\mathbb{Z}_2$. Since, in mod 2 arithmetic, the last equation is the sum of the first three equations, it can be erased. The remaining three equation can be solved in terms of x, y and u as

$$\begin{aligned} z &= x + y \\ v &= x + u \\ w &= y + u \end{aligned}$$

where $x, y, u \in \mathbb{Z}_2$ are arbitrary. Hence, the set of 1-cycles consists of

$$Z_1 = \Big\{ x \cdot \left(\overline{AB} + \overline{AD} + \overline{BD}\right) + y \cdot \left(\overline{AC} + \overline{AD} + \overline{CD}\right) + u \cdot \left(\overline{BC} + \overline{BD} + \overline{CD}\right) \quad \text{where } x, y, u \in \mathbb{Z}_2 \Big\}.$$

We are now ready to find the 1-boundaries among the 1-cycles described above in the set Z_1. Observe that the three brackets in the expression for an 1-cycle are all boundaries of one of the triangle faces of the tetrahedron. More precisely,

$$\begin{aligned} \partial_2 \left(\triangle ABD\right) &= \overline{AB} + \overline{AD} + \overline{BD} \\ \partial_2 \left(\triangle ACD\right) &= \overline{AC} + \overline{AD} + \overline{CD} \\ \partial_2 \left(\triangle BCD\right) &= \overline{BC} + \overline{BD} + \overline{CD}. \end{aligned}$$

Therefore, for any $x, y, u \in \mathbb{Z}_2$, we have

$$\begin{aligned} \partial_2 \left(x \cdot \triangle ABD + y \cdot \triangle ACD + u \cdot \triangle BCD\right) =& x \cdot \left(\overline{AB} + \overline{AD} + \overline{BD}\right) \\ &+ y \cdot \left(\overline{AC} + \overline{AD} + \overline{CD}\right) \\ &+ u \cdot \left(\overline{BC} + \overline{BD} + \overline{CD}\right), \end{aligned}$$

which shows that every 1-cycle is at the same time an 1-boundary. In other words, $Z_1 = B_1$, so that the homology group

$$H_1 = \{0\}$$

is trivial, and the Betti number $\beta_1 = 0$.

It remains to determine homology in dimension zero. Since all 0-chains are 0-cycles, we have

$$Z_1 = C_1 = \Big\{ q \cdot A + r \cdot B + s \cdot C + t \cdot D \quad \text{where } q, r, s, t \in \mathbb{Z}_2 \Big\}.$$

The number of 0-cycles is $2^{n_0} = 2^4 = 16$. It is sufficient to determine which of these 0-cycles are 0-boundaries. Since the 1-boundary operator acts on the edges of the tetrahedron, and each of them has two end-points, it turns out that in the boundary of any 1-chain, there is an even number of points in total. This means that among elements in Z_1, the 1-boundaries are determined by the condition that

$$q + r + s + t = 0$$

in mod 2 arithmetic. This means that the number of 0-boundaries is $2^{4-1} = 8$, as the last coefficient is always determined by the other three according to the condition above. Therefore, the Betti number equals $\beta_0 = 4 - 3 = 1$.

In conclusion, we have determined the Betti numbers

$$\beta_0 = 1,$$
$$\beta_q = 0, \text{ for } q > 0,$$

for homology over $\mathbb{Z}_2$.

More generally, the homology over $\mathbb{Z}_2$ of any p-simplex, viewed as a simplicial complex with all of its subsimplices, exhibits a similar pattern of Betti numbers. The only non-zero Betti number is $\beta_0 = 1$. There is a geometric reason underlying this result. Any p-simplex, as for instance the tetrahedron, can be contracted into a point, without gluing or cutting. Hence, the topological invariants of a p-simplex (or any other convex set), are those of a single point. But for a single point, which is a 0-simplex, the top dimension is zero. And in dimension zero, the point represents a cycle, which is not a boundary. Thus, $\beta_0 = 1$, and $\beta_q = 0$ for all $q > 0$, for homology over $\mathbb{Z}_2$ of any p-simplex.

4.11 Example 2: Homology of a 2-sphere

The homology of a 2-sphere is calculated using its triangulation. The result is independent of the choice of triangulation and we use the one already given in Fig. 7. This triangulation is a simplicial complex very close to the simplicial complex of the tetrahedron in the previous example. The only difference is that the 3-simplex given by the whole tetrahedron is removed from the simplicial complex.

Hence, all the calculations in the previous example apply here, except for the homology groups in the top dimension. In the case of 2-sphere the top dimension is two, because there are no 3-simplices in the simplicial complex. However, the 2-cycles remain the same as in the case of the tetrahedron. Thus, the number of 2-cycles is 2. But since there are no 3-simplices, there are no 2-boundaries, so that the homology group is

$$H_2 = Z_2,$$

and the Betti number is $\beta_2 = 1$. The other homology groups and the Betti numnbers are exactly the same as in the example of the tetrahedron. Thus, the Betti numbers of the homology over $\mathbb{Z}_2$ of the 2-sphere are

$$\beta_0 = \beta_2 = 1,$$

$$\beta_p = 0 \text{ for all } p \neq 0, 2.$$

The same argument, relying on the homology of an $(n+1)$-simplex, implies that the Betti numbers for the n-sphere, i.e., the n-dimensional sphere in the $(n+1)$-dimensional space, are given by

$$\beta_0 = \beta_n = 1,$$

$$\beta_p = 0 \text{ for all } p \neq 0, n.$$

for homology over $\mathbb{Z}_2$.

5 A running example

In the real world, a software graph will never be a simple graph. The vertices of a software graph may represent different things: classes, objects, functions, modules, components, software units etc. The edges of a software graph represent communication or calls between these parts of the software. In any case, the software graph is always directed, as the calls between parts of the software are directed from one part to the other. The software graph is often weighted or with multiple edges, because some calls between parts of the software are more frequent than others. This is encapsulated in the graph by assigning weights to the edges or allowing multiple edges for multiple calls. Even the loops may appear in a software graph, as they would represent recursive calls to the same part of software.

On the other hand, the topological algorithm presented in Sect. 4 requires a simple graph as input, works over the field $\mathbb{Z}_2 = \{0, 1\}$ of two elements, and defines simplices as complete subgraphs. These simplifications are necessary to make the presentation of the basic concepts more accessible to students. Nevertheless, it still captures certain amount of topological insight in higher dimensional structure of software. The topological techniques of similar nature can be applied to more general settings, such as the cases of directed and weighted graphs and working over other field and rings than $\mathbb{Z}_2$. The reference for these techniques could be any textbook on algebraic topology such as [29].

In the real world software is certainly a limitation, although there is still some topological insight gained from the presented algorithm.

5.1 Input to the algorithm

The running example considered now is given by the graph given in Fig. 16. It is a small graph consisting of only four vertices, denoted in the figure by numbers 1, 2, 3 and 4, but it is not a simple graph. There are multiple edges and it is directed. Hence, the input to the running example is just a list of edges of a software graph. The first two numbers in each line represent an edge between the vertices labeled by these numbers. The direction of the edge is the order of numbers. It is also possible that the edges are weighted, which is indicated in the list by the third number in each row. In the running example, all the weights are set to 1. For the running example in Fig. 16, the list of edges is given by

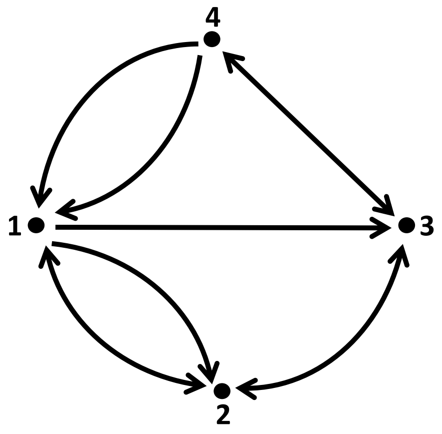


Fig. 16: The graph of the running example.

```
1 2 1
2 3 1
1 3 1
2 1 1
1 2 1
3 2 1
3 4 1
4 1 1
4 1 1
4 3 1
```

However, since the topological algorithm that we introduced in Sect. 4 requires a simple graph as input, the list of edges must be cleaned up. The multiple edges must be replaced by a single undirected copy of an edge, and the weights must be erased. Hence, the input to the topological algorithm for the running example is the list

$$S_1 = \begin{bmatrix} 2\,3 \\ 1\,3 \\ 2\,1 \\ 4\,1 \\ 4\,3 \end{bmatrix}.$$

where multiple edges and weights are removed. The simple graph obtained by this procedure is given in Fig. 17.

5.2 Step 0 – adjacency matrix

The step 0 of the algorithm is to write down the adjacency matrix of the given simple graph. It will be used repeatedly in the following steps as it encodes infor-

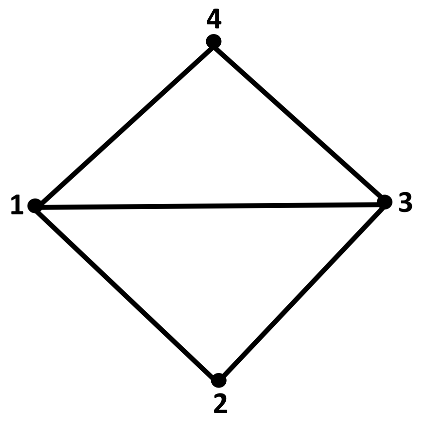


Fig. 17: The simple graph obtained in the running example.

mation about neighboring vertices. This information is required in constructing the simplicial complex and boundary operators introduced in Sect. 4, more precisely, in Sect. 4.1, Sect. 4.3 and Sect. 4.5.

Recall that the adjacency matrix of a graph is a square matrix with the number of rows and columns equal to the number of vertices of the graph. Let n_0 denote the number of vertices in the graph, so that the adjacency matrix is an $n_0 \times n_0$ matrix. The order of vertices must be chosen and fixed. Then, the elements of the adjacency matrix are determined as follows.

There is 1 at the crossing of the i-th row and the j-th column, if the i-th and the j-th vertex are connected by an edge,

There is 0 at the crossing of the i-th row and the j-th column, otherwise.

Observe that we consider a simple graph. Hence, the adjacency matrix is symmetric, because the edges are undirected. It contains only ones and zeroes, because there are neither multiple edges nor weights. The diagonal entries are zero, because there are no loops.

In the running example $n_0 = 4$, and the order of vertices is already fixed by the labels 1, 2, 3, 4. The adjacency matrix is

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

Observe that A has all the properties mentioned for the adjacency matrix of a simple graph.

5.3 Step 1 – incidence matrix

The next step is to determine the incidence matrix of a given graph. It is nothing else than the matrix of the boundary operator in dimension one, as defined in Sect. 4.6.

Recall that the incidence matrix of a graph is a matrix with the number of rows equal to the number n_0 of vertices, and the number of columns equal to the number n_1 of edges. Thus, the incidence matrix is an $n_0 \times n_1$ matrix. The order of vertices and edges must be fixed once and for all. Then, the entries of the incidence matrix are determined as follows.

> There is 1 at the crossing of the i-th row and the j-th column, if the i-th vertex belongs to the j-th edge,
> There is 0 at the crossing of the i-th row and the j-th column, otherwise.

Since every edge has two vertices as its end-points, each column of the incidence matrix contains exactly two entries equal to 1, and the rest are 0.

In the running example, there are $n_0 = 4$ vertices, ordered by their labels, and $n_1 = 5$ edges, ordered as in the given list S_1 of edges. The order of edges may be chosen differently. The most canonical choice would be to order the edges in the lexicographical order with respect to the order of edges, but this is not so important in this small example. With the fixed order of vertices and edges as above in S_1, the incidence matrix in the running example is

$$\partial_1 = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} .$$

Observe that the incidence matrix is denoted by ∂_1, as the 1-boundary operator. This is not a coincidence. The incidence matrix is precisely the matrix of the 1-boundary operator written with the fixed order of vertices, which are 0-simplices, and edges, which are 1-simplices of the simplicial complex assigned to the graph. The point is that the j-th column, which represents the j-th edge, i.e., j-th 1-simplex, contains 1 precisely in the rows representing its end-points, i.e., its 0-simplex faces. Thus, the incidence matrix is indeed equal to the matrix of the 1-boundary operator.

For convenience of the reader, following the suggestion of the referee, we draw here the table from which the matrix of the boundary operator ∂_1 is determined

$\partial_1 \sim$

	2, 3	1, 3	2, 1	4, 1	4, 3
1	×	√	√	√	×
2	√	×	√	×	×
3	√	√	×	×	√
4	×	×	×	√	√

The rows are labeled with 0-simplices (i.e., vertices) and columns are labeled with 1-simplices (i.e., edges). The signs √ and × in the table represent whether the vertex of a given row is the end-point of the edge of a given column, or not. In the matrix, these signs become ones and zeroes.

Similar table for the boundary operator ∂_2 is as follows.

$$\partial_2 \sim \begin{array}{c||c|c} & 2,3,1 & 1,3,4 \\ \hline\hline 2,3 & \surd & \times \\ \hline 1,3 & \surd & \surd \\ \hline 2,1 & \surd & \times \\ \hline 4,1 & \times & \surd \\ \hline 4,3 & \times & \surd \end{array}$$

In this table, the rows are labeled with 1-simplices (i.e., edges) and columns are labeled with 2-simplices (i.e., triangles), and signs $\surd$ and $\times$ represent whether a given edge is the side of a given triangle.

In this way, it becomes clear how the matrices of the boundary operator are determined from the knowledge of boundary faces of every simplex in a simplicial complex.

5.4 Step 2 – higher differentials

The algorithm proceeds iteratively by computing step-by-step the higher differentials of the simplicial complex, that is, the matrices of the p-boundary operators for $p \geq 2$.

There are a few sloppy places in applying this procedure. First of all, one should make sure that the list S_p of p-simplices does not contain several copies of the same p-simplex. Hence, it is desirable to fix an order of vertex labels (increasing or decreasing) when making the list in order to avoid double simplices in S_p.

Another issue is that adding a column in the matrix ∂_p of p-boundary operator requires the knowledge of the row numbers corresponding to all of its faces. Perhaps a good idea is to make the list S_p in such a way that the order of simplices is easy to handle and allows fast search.

Recall from Sect. 4.6 that the matrix of the p-boundary operator ∂_p is a matrix with n_{p-1} rows and n_p columns, where n_k is the number of k-simplices, i.e., the number of complete subgraphs with $k+1$ vertices. The entries in the matrix are determined as follows.

> There is 1 at the crossing of the i-th row and the j-th column, if the i-th $(p-1)$-simplex, i.e., the i-th complete subgraph with p vertices, is a $(p-1)$-simplex face of the j-th p-simplex, i.e., a subgraph of the the j-th complete subgraph with $p+1$ vertices,
>
> There is 0 at the crossing of the i-th row and the j-th column, otherwise.

Observe that this requires a fixed ordering of complete subgraphs with a given number of vertices. Such complete subgraphs are determined iteratively, and their order is fixed along the way. Once it is fixed, the order should not be changed. In particular, the numbering of complete subgraphs with p vertices should be the same in the construction of matrices of boundary operators ∂_p and ∂_{p-1}. In this way, we obtain at the end the list of all simplices in the simplicial complex, as in Sect. 4.1.

The problem of finding all complete subgraphs in an arbitrary simple graph is computationally highly demanding. However, the software graphs are not, and should not, be arbitrary. Larger complete subgraphs in a software graph would indicate a high level of communication between many different parts of software, which is not appropriate for design, testing, verification and maintenance of software. Software design principles specify that such communication should be avoided. Therefore, in the special case of software graphs, the search for complete subgraphs is feasible.

The list S_1 of edges may and will serve as the list of 1-simplices with the fixed order, as above in the construction of the incidence matrix ∂_1. The iterative computation of differentials requires to keep track of the p-simplices in each dimension p, and their fixed order. The list S_p of p-simplices is a matrix with n_p rows and $p+1$ columns, where n_p is the number of complete subgraphs with $p+1$ vertices (i.e. p-simplices) such that each row contains $p+1$ integers which are labels of points that form a p-simplex.

As the first iterative step, one should determine the matrix ∂_2 and the list S_2 of complete subgraphs with 3 vertices (i.e. 2-simplices) from the matrix ∂_1 and the list S_1 of edges (i.e. 1-simplices) using the adjacency matrix A. This is done as follows.

1. Consider the first previously unused row in the list S_1 of edges (1-simplices)
2. Check in the adjacency matrix A if all the vertices of the 1-simplex defined by that row have a vertex as a common neighbor, that is, if all the vertices of the 1-simplex are connected by an edge to the same vertex of a graph.
3. For each common neighbor determined in Step 2, adding such a common neighbor to the 1-simplex gives a 2-simplex (a triangle in the graph), which is stored in the list S_2 of 2-simplices.
4. For each 2-simplex determined in Step 2, add a column on the right-hand side of the 2-boundary operator matrix ∂_2 which contains ones in the rows of edges of that 2-simplex.
5. Go back to Step 1, unless there are no more unused rows in S_1.

In this way, adding all possible neighbors to the 1-simplices, we obtain the list S_2, and the order of the list is fixed once and for all. At the same time, for each 2-simplex, a column on the right-hand side of the 2-boundary operator matrix ∂_2 is added. The outcome of the algorithm is the ordered list S_2 and the matrix ∂_2 obtained simultaneously.

In the running example the list of 2-simplices (i.e., triangles in the graph) are the following

$$S_2 = \begin{bmatrix} 2 & 3 & 1 \\ 1 & 3 & 4 \end{bmatrix}$$

and the 2-boundary operator matrix is

$$\partial_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} .$$

There are only two triangles in the graph, so that the list S_2 has two rows and the matrix ∂_2 has two columns.

In general, how to determine the p-boundary operator matrix ∂_p and the list S_p of p-simplices from the $(p-1)$-boundary operator matrix ∂_{p-1} and the list S_{p-1} of $(p-1)$-simplices using the adjacency matrix A? In precisely the same way!

Go through the rows of the list S_{p-1} of $(p-1)$-simplices, and for each row of S_{p-1}, check in the adjacency matrix A if all the vertices of the $(p-1)$-simplex defined by that row have a common neighbor. Adding such common neighbor to the $(p-1)$-simplex gives a p-simplex, which should be stored in the list S_p of p-simplices. When a p-simplex is detected, at the same time, add a column in the p-boundary operator matrix ∂_p. This added column contains ones in the rows of $(p-1)$-simplex faces of that p-simplex.

In the running example, as small as it is, there are no 3-simplices (i.e. comlete subgraphs with four vertices). Hence, the list S_3 is empty, and there is no matrix ∂_3.

5.5 Step 3 – rank of a matrix over $\mathbb{Z}_2$

All the information required for the calculation of homology groups and their ranks, introduced in Sect. 4.9, are encoded in the matrix ∂_p of the p-boundary operator for all p, which are defined in Sect. 4.6. In particular, the rank of the matrices ∂_p contains the information regarding the rank of the cycle and boundary groups, as defined in Sect. 4.8, which are required for computing the Betti numbers, i.e., the rank of homology groups, as in Sect. 4.9.

This information can be read off from a normal form of a matrix. The reduction of a matrix to a normal form can be made using the well-known Gauss elimination method. The students learn this algorithm in the basic linear algebra course, and over $\mathbb{Z}_2$ it is even simpler.

The Gauss elimination over $\mathbb{Z}_2$ consists of the following steps, where the input is any matrix with entries in $\mathbb{Z}_2 = \{0, 1\}$.

1. Set the considered part of the matrix to be the whole matrix.
2. Find a place in the considered part of the matrix containing 1. If it does not exist, the matrix is already in a normal form, exit the algorithm.
3. Exchange two rows and two columns, so that this 1 appears in the upper-left corner of the considered part of the matrix.
4. Find 1's in the first column of the considered part of the matrix.

5. Add the first row of the considered part of the matrix to those rows that contain 1's in the first column of the considered part of the matrix. The addition is made mod 2, so that these 1's are canceled.
6. Change the considered part of the matrix by removing its first row and its first column. Go through steps 2–5 with this new considered part of the matrix.

The outcome of the Gauss elimination is the matrix of the form

$$M = \begin{bmatrix} 1 & * & \dots & * & * & \dots & * \\ 0 & 1 & \ddots & \vdots & \vdots & & \vdots \\ \vdots & \ddots & \ddots & * & \vdots & & \vdots \\ 0 & \dots & 0 & 1 & * & \dots & * \\ 0 & \dots & \dots & 0 & 0 & \dots & 0 \\ \vdots & & & & & & \vdots \\ 0 & \dots & \dots & 0 & 0 & \dots & 0 \end{bmatrix}.$$

The rank of the matrix M is defined as the number of 1's along the diagonal. The defect of the matrix M is the number of remaining columns. Although we did not define the rank and defect in general, note that the rank and defect of any matrix is equal to the rank and defect of its normal form as defined for the matrix M.

It turns out that in the case of the matrix ∂_p of the p-boundary operator

the rank of ∂_p equals the rank b_{p-1} of the boundary group B_{p-1}, as defined in Sect. 4.8,
the defect of ∂_p equals the rank z_p of the cycle group Z_p, as defined in Sect. 4.8.

Thus, the rank and defect of ∂_p for all p provides all the necessary information to compute the Betti numbers of the given graph, that is, the Betti numbers of the considered software.

In the running example, the Gauss elimination of the matrix ∂_1 is performed as follows. In the first step we move the 1 at the crossing of the second row and first column to the upper-left corner. This is achieved by replacing the first two rows of the matrix. In the second step, the first row is added to the third row to cancel the 1 in the first column.

$$\partial_1 = \begin{bmatrix} 0&1&1&1&0 \\ 1&0&1&0&0 \\ 1&1&0&0&1 \\ 0&0&0&1&1 \end{bmatrix} \sim \begin{bmatrix} 1&0&1&0&0 \\ 0&1&1&1&0 \\ 1&1&0&0&1 \\ 0&0&0&1&1 \end{bmatrix} \sim \begin{bmatrix} 1&0&1&0&0 \\ 0&1&1&1&0 \\ 0&1&1&0&1 \\ 0&0&0&1&1 \end{bmatrix}.$$

Now the considered part of the matrix is obtained by removing the first row and the first column. In the considered part of the matrix, there is 1 already in the upper-left corner (which is the crossing of the second row and the second column of the whole matrix), so that no replacing of rows and columns is necessary. Thus, the next step is to add the second row to the third row, which cancels the 1 in the second column below the upper-left corner of the considered part of the matrix. We obtain

$$\partial_1 \sim \begin{bmatrix} 1\,0\,1\,0\,0 \\ 0\,1\,1\,1\,0 \\ 0\,0\,0\,1\,1 \\ 0\,0\,0\,1\,1 \end{bmatrix}.$$

Now the considered part of the matrix is obtained by removing again the first row and the first column. In other words, we consider the matrix above with the first two rows and the first two columns removed. In the remaining part of the matrix, there is 0 in the upper-left corner (which is the crossing of the third row and the third column). Hence, we must replace the third and the fourth columns of the matrix, and then add the third row to the fourth row to cancel the 1 below.

$$\partial_1 \sim \begin{bmatrix} 1\,0\,0\,1\,0 \\ 0\,1\,1\,1\,0 \\ 0\,0\,1\,0\,1 \\ 0\,0\,1\,0\,1 \end{bmatrix} \sim \begin{bmatrix} 1\,0\,0\,1\,0 \\ 0\,1\,1\,1\,0 \\ 0\,0\,1\,0\,1 \\ 0\,0\,0\,0\,0 \end{bmatrix}.$$

The last line has become zero, so we are done. The final matrix is in the normal form, and we can read off the rank and defect of ∂_1. The rank is 3, because there are 3 ones along the diagonal, and the defect is 2, because there are two columns remaining. Thus,

$$b_0 = 3,$$
$$z_1 = 2,$$

is the rank of the group B_0 of 0-boundaries, and the rank of the group Z_1 of 1-cycles, respectively.

Similarly, we perform the Gauss elimination method for the matrix ∂_2 of the 2-boundary operator in the running example. In the first step, the first row is added to the second and the third row to cancel the 1's below the upper-left corner. Then, the considered part of the matrix is obtained by removing the first row and the second column. The 1 at the crossing of the second row and the second column is used to cancel all the 1's below it by adding the second row to fourth and fifth row.

$$\partial_2 = \begin{bmatrix} 1\,0 \\ 1\,1 \\ 1\,0 \\ 0\,1 \\ 0\,1 \end{bmatrix} \sim \begin{bmatrix} 1\,0 \\ 0\,1 \\ 0\,0 \\ 0\,1 \\ 0\,1 \end{bmatrix} \sim \begin{bmatrix} 1\,0 \\ 0\,1 \\ 0\,0 \\ 0\,0 \\ 0\,0 \end{bmatrix}$$

The last matrix is the normal form of ∂_2. Hence, the rank and defect of ∂_2 is read off that matrix. It gives

$$b_1 = 2,$$
$$z_2 = 0,$$

is the rank of the group B_1 of 1-boundaries, and the rank of the group Z_2 of 2-cycles, respectively.

5.6 Final step – Betti numbers in homology

Finally, all the acquired information is combined to obtain the Betti numbers in homology of the simplicial complex assigned to a software graph, as in Sect. 4.9. In general,

$z_0 = n_0$ as always,
b_{p-1} and z_p are obtained from ∂_p for $1 \leq p \leq t$,
$b_t = 0$,

where t is the top dimension such that there are no $(t+1)$-simplices from which the t-boundaries could come from. From these values, as in Sect. 4.9, the Betti numbers are

$$\beta_p = z_p - b_p$$

for all $p = 0, 1, \ldots, t$.

In the case of the running example, we have calculated the following

$z_0 = n_0 = 4$ as always,
$b_0 = 3$ is obtained from ∂_1,
$z_1 = 2$ is obtained from ∂_1,
$b_1 = 2$ is obtained from ∂_2,
$z_2 = 0$ is obtained from ∂_2,
$b_2 = 0$ because 2 is the top dimension, as there are no 3-simplices.

Therefore, the Betti numbers are

$$\beta_0 = z_0 - b_0 = 4 - 3 = 1,$$
$$\beta_1 = z_1 - b_1 = 2 - 2 = 0,$$
$$\beta_2 = z_2 - b_2 = 0 - 0 = 0.$$

6 Exercise

The goal of the following exercise is to write a code performing the algorithm described in the previous sections. The idea is that writing the code will make the abstract topological notions more familiar to students, and testing the code on different types of graphs will give them the flavor of the difficulties in terms of computational power and sustainability of the algorithm. It is important that the solution is fast enough to handle software graphs, which do not contain too large complete subgraphs.

Exercise: Given a software graph in the form described below, write a code in the programming language of your choice, which computes the Betti numbers, i.e., the ranks of homology groups, for that software.

The input to your code is a software graph given as a list of edges. The vertices are labeled by positive integers up to n_0, which denotes the number of vertices. The edges are pairs of such integers. Let n_1 denote the number of edges. The list of edges is a matrix (an array) with three columns and n_1 rows, where each row represents an edge:

The first integer in a row is the starting vertex of an edge.
The second integer in a row is the ending vertex of an edge.
The third number in a row is the weight of an edge.

For our task the directions of the edges are ignored, multiple edges are ignored and the weights are ignored. Hence, the code should first clean up the list of edges by removing the weights, i.e., erasing the third column, and removing the duplicate edges in the list. Note that the order of vertices of an edge is irrelevant as the graph is unordered.

The steps of the algorithm described in Sect. 5 should be implemented in the code as the solution of the exercise. These steps lead to the solution.

Step 0 Write the code which transforms the cleaned up input list of edges into the adjacency matrix A of a given graph.

Step 1 Write the code which transforms the cleaned up input list of edges into the incidence matrix of a given graph, which is at the same time the matrix ∂_1 of the 1-boundary operator.

Step 2 Write the code which computes iteratively the matrix of higher-dimensional boundary operators ∂_p and at the same time the ordered list S_p of p-simplices.

Step 3 Write the code which implements the Gauss elimination algorithm and determines the normal form of a given matrix. Apply it to the matrices of boundary operators ∂_p.

Final step Write the code that reads off the rank and defect from a normal form of a matrix. Apply it to the normal forms of the boundary operators ∂_p and compute the Betti numbers.

The code that solves the exercise can be tested on any simple graph. However, for arbitrary graphs there is no solution that is fast enough to determine the Betti numbers for the simplicial complex assigned to a graph as in the previous sections. The problem occurs if the graph is highly connected in terms of complete subgraphs. The solution code should work fast enough to handle the software graphs, in which there are usually only smaller complete subgraphs.

At the summer school, the attendees were given a sample software graph as the test input. It was a software graph with 3833 vertices and 17602 edges. The top dimension of the graph turned out to be only $t = 3$, so that there are no complete subgraphs with more than 4 vertices. The Betti numbers are

$$\begin{aligned}\beta_0 &= 7,\\ \beta_1 &= 12591,\\ \beta_2 &= 1995,\\ \beta_3 &= 46.\end{aligned}$$

The interested reader who solved the exercise is urged to either contact the authors for the sample graph input to test their solution, or test the solution on any other software graph available.

7 Conclusion

Software is the main technology driving digitalization in many domains as support in decision-making processes. It abstracts the physical world and manages such abstract resources and it implements machine learning and artificial intelligence algorithms to provide analysis of big data. The way how we engineer software solutions has a significant impact on addressing sustainability goals in the application domain where software is implemented but also within the software engineering industry. Engineering sustainable software is becoming crucial for future technology advancements. The main obstacle in engineering sustainable software is the human ability to engineer complex systems.

In this lecture, we define complex systems and specific challenges of modeling modern software systems and their consequences on sustainable software behavior. We explain the problem of modeling complex software behavior from the perspective of modeling local system properties that are measured on software parts and global system properties that are measured in the system operation. Furthermore, we introduce the software structure as one of the key instruments we model relations between local and global system properties and explain its graph representation.

Finally, we introduce students to Topological Data Analysis (TDA) as an analysis tool that may be very useful in modeling complex systems as a complementary tool to various existing graph algorithms. TDA is capable of describing the topological space of structure that may introduce an additional useful dimension to explain algorithmic decisions in various applications. Here we introduce the key concepts of TDA and guide students on how to implement these abstract topological notions.

In our running case, we showed how TDA may be used in the analysis of software structures with the help of TDA aiming to address concerns of sustainable software evolution. The results of our case of using TDA to model software structures in evolution are presented in [37]. By experimenting with different graphs the students may understand the difficulties in terms of computational power and sustainability of the algorithm by testing the code on different types of graphs, and how this implementation may impact sustainable software structures.

Acknowledgements This paper acknowledges the support of the Erasmus+ Key Action 2 (Strategic partnership for higher education) project No. 2020-1-PT01-KA203-078646: "SusTrainable – Promoting Sustainability as a Fundamental Driver in Software Development Training and Education" and the support of the Croatian Science Foundation under the projects HRZZ-IP-2019-04-4216 and HRZZ-IP-2022-10-4615. The information and views set out in this paper are those of the author(s) and do not necessarily reflect the official opinion of the European Union. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein.

References

1. Aliance Next Generation Mobile Networks (NGMN): Green future networks: Network energy efficiency. (2021), https://www.ngmn.org/publications/green-future-networks-network-energy-efficiency.html
2. Allili, M., Kaczynski, T., Landi, C., Masoni, F.: Acyclic partial matchings for multidimensional persistence: Algorithm and combinatorial interpretation. J. Math. Imaging Vis. **61**(2), 174–192 (2019)
3. Bendich, P., Edelsbrunner, H., Kerber, M.: Computing robustness and persistence for images. IEEE Trans. Vis. Comput. Graph. **16**(6), 1251–1260 (2010)
4. Callahan, D., Carle, A., Hall, M.W., Kennedy, K.: Constructing the procedure call multigraph. IEEE Transactions on Software Engineering **16**(4), 483–487 (1990).
5. Cang, Z., Mu, L., Wu, K., Opron, K., Xia, K., Wei, G.W.: A topological approach for protein classification. Computational and Mathematical Biophysics **3**(1) (2015)
6. Carlsson, G.: Topology and data. Bull. Amer. Math. Soc. (N.S.) **46**(2), 255–308 (2009)
7. Carlsson, G., Vejdemo-Johansson, M.: Topological data analysis with applications. Cambridge University Press, Cambridge (2022)
8. Choudhary, A., Kerber, M., Raghvendra, S.: Improved approximate rips filtrations with shifted integer lattices and cubical complexes. J. Appl. Comput. Topol. **5**(3), 425–458 (2021)
9. Dey, T.K., Mandal, S., Mukherjee, S.: Gene expression data classification using topology and machine learning models. BMC Bioinform. **22-S** (Suppl. 10), Article no. 627 (2021)
10. Edelsbrunner, H., Harer, J.: Computational Topology – an Introduction. American Mathematical Society (2010)
11. Edelsbrunner, H., Morozov, D.: Persistent homology: theory and practice. In: Proceedings of the European Congress of Mathematics 2012, pp. 31–50. EMS Press, Berlin (2014)
12. Fremerey, C., Mueller, M., Clausen, M.: Towards bridging the gap between sheet music and audio. In: Knowledge representation for intelligent music processing. Dagstuhl Seminar Proceedings (DagSemProc), vol. 9051, pp. 1–11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2009)
13. Galinac Grbac, T.: The role of functional programming in management and orchestration of virtualized network resources. Part I. System structure for complex systems and design principles. In: Composability, Comprehensibility and Correctness of Working Software, 7th Winter School. Revised Selected Papers, Lecture Notes in Computer Science vol. 11916, Springer, Cham (2023), to appear. Available at https://arxiv.org/abs/2107.12136
14. Galinac Grbac, T., Domazet, N.: The role of functional programming in management and orchestration of virtualized network resources. Part II. Network evolution and design principles. In: Composability, Comprehensibility and Correctness of Working Software, 8th Summer School. Revised Selected Papers, Lecture Notes in Computer Science vol. 11950, Springer, Cham (2023), to appear. Available at https://arxiv.org/abs/2107.12227
15. Gasarch, W., Fasy, B.T., Wang, B.: Open problems in computational topology. SIGACT News **48**(3), 32–36 (2017)
16. Giray, G., Bennin, K.E., Köksal, Ö, Babur, Ö, Tekinerdogan, B.: On the use of deep learning in software defect prediction. Journal of Systems and Software **195**, Article no. 111537 (2023)

17. Hatcher, A.: Algebraic topology. Cambridge University Press, Cambridge (2002)
18. Hilty, L.M., Aebischer, B.: ICT for sustainability: An emerging research field. In: Hilty, L.M., Aebischer, B. (eds.) ICT Innovations for Sustainability. pp. 3–36. Springer International Publishing, Cham (2015)
19. Hilty, L.M., Arnfalk, P., Erdmann, L., Goodman, J., Lehmann, M., Wäger, P.A.: The relevance of information and communication technologies for environmental sustainability – a prospective simulation study. Environmental Modelling & Software **21**(11), 1618–1629 (2006)
20. International Organization for Standardization (ISO), International Electrotechnical Commission (IEC): ISO/IEC 25002:2024 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Quality model overview and usage. International standard, 1st edition (2024).
21. Kang, L., Xu, B., Morozov, D.: Evaluating state space discovery by persistent cohomology in the spatial representation system. Frontiers Comput. Neurosci. **15**, Article no. 616748 (2021)
22. Kim, H.S., Yi, C., Kim, Y., Park, U., Kook, W., Oh, B., Kim, H., Park, T.: Topological data analysis can extract sub-groups with high incidence rates of type 2 diabetes. Int. J. Data Min. Bioinform. **22**(1), 44–60 (2019)
23. Lago, P., Koçak, S.A., Crnković, I., Penzenstadler, B.: Framing sustainability as a property of software quality. Commun. ACM **58**(10), 70–78 (2015)
24. Lago, P., Meyer, N., Morisio, M., Müller, H.A., Scanniello, G.: Leveraging "energy efficiency to software users": Summary of the second greens workshop, at ICSE 2013. SIGSOFT Softw. Eng. Notes **39**(1), 36–38 (2014)
25. Leykam, D., Angelakis, D.G.: Topological data analysis and machine learning. Advances in Physics: X **8**(1), Art. no. 2202331, 24 pages (2023)
26. Malott, N.O., Chen, S., Wilsey, P.A.: A survey on the high-performance computation of persistent homology. IEEE Trans. Knowl. Data Eng. **35**(5), 4466–4484 (2023)
27. Mauša, G., Galinac Grbac, T.: Co-evolutionary multi-population genetic programming for classification in software defect prediction: An empirical case study. Appl. Soft Comput. **55**, 331–351 (2017)
28. Munch, E.: A user's guide to topological data analysis. Journal of Learning Analytics **4**(2), 47–61 (2017)
29. Munkres, J.R.: Elements of algebraic topology. Addison-Wesley Publishing Company, Menlo Park, CA (1984)
30. Munkres, J.R.: Topology. Prentice Hall, Inc., Upper Saddle River, NJ (2000)
31. Pandey, S.K., Mishra, R.B., Tripathi, A.K.: Machine learning based methods for software fault prediction: A survey. Expert Systems with Applications **172**, Article no. 114595 (2021)
32. Penzenstadler, B., Raturi, A., Richardson, D., Tomlinson, B.: Safety, security, now sustainability: The nonfunctional requirement for the 21st century. IEEE Software **31**(3), 40–47 (2014)
33. Petrić, J., Galinac Grbac, T.: Software structure evolution and relation to system defectiveness. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14. pp. 34:1–34:10. ACM (2014)
34. Petrić, J., Galinac Grbac, T., Dubravac, M.: Processing and data collection of program structures in open source repositories. In: Proceedings of the 3rd Workshop on Software Quality Analysis, Monitoring, Improvement and Applications (SQAMIA 2014). CEUR Workshop Proceedings, vol. 1266, pp. 57–66. CEUR (2014)
35. Pita Costa, J., Škraba, P.: A topological data analysis approach to the epidemiology of influenza. In: SIKDD15 Conference Proceedings (2015)

36. Pita Costa, J., Galinac Grbac, T.: The topological data analysis of time series failure data in software evolution. In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion. pp. 25–30. ICPE '17 Companion, ACM, New York, NY, USA (2017).
37. Puh E., Galinac Grbac, T., Grbac, N.: Preliminary study of higher dimensional software structures. In: Proceedings of the 10th Workshop on Software Quality Analysis, Monitoring, Improvement and Applications (SQAMIA 2023). CEUR Workshop Proceedings, vol. 3588, pp. 13–25. CEUR (2023)
38. Valverde, S., Solé, R.V.: Network motifs in computational graphs: a case study in software architecture. Physical review. E, Statistical, nonlinear, and soft matter physics **72 2 Pt 2**, 026107–1, 026107–8 (2005)
39. Vranković, A., Galinac Grbac, T., Car, Ž.: Software structure evolution and relation to subgraph defectiveness. IET Softw. **13**(5), 355–367 (2019)