jHoles: a tool for understanding biological complex networks via clique weight rank persistent homology

Jacopo Binchi[a], Emanuela Merelli[a], Matteo Rucco[a] [1]

[a] *School of Science and Technology, Computer Science Division, University of Camerino, Camerino, Italy*

Giovanni Petri[b], Francesco Vaccarino[b,c]

[b] *ISI Foundation, Torino, Italy.* [c] *Dipartimento di Scienze Matematiche, Politecnico di Torino, Torino, Italy*

Abstract

Complex networks equipped with topological data analysis are one of the promising tools in the study of biological systems (e.g. evolution dynamics, brain correlation, breast cancer diagnosis, etc...). In this paper, we propose *jHoles*, a new version of *Holes*, an algorithms based on persistent homology for studying the connectivity features of complex networks. *jHoles* fills the lack of an efficient implementation of the filtering process for clique weight rank homology. We will give a brief overview of *Holes*, a more detailed description of *jHoles* algorithm, its implementation and the problem of clique weight rank homology. We present a biological case study showing how the connectivity of epidermal cells changes in response to a tumor presence. The biological network has been derived from the proliferative, differentiated and stratum corneum compartments, and *jHoles* used for studying variation of the connectivity.

Keywords: Complex networks, Biological networks, Tumor diagnosys, Computational topology, Betti number

1 Introduction

Complex networks are one of the most used tool for studying complex systems. In particular, weighted networks are becoming a more and more important tool to detect either the presence and the intensity of relations among

[1] Email: matteo.rucco@unicam.it - Corresponding author

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

groups of nodes in a network. Topological data analysis (TDA for short) is a subarea of computational topology that develops topological techniques for robust analysis of scientific data. Topology is the branch of geometry that studies shapes, it classifies objects according to properties that do not change under certain feasible transformations to capture more qualitative information about shapes. In mathematics (especially algebraic topology and abstract algebra), in algebraic topology homology is a general procedure to associate a sequence of abelian groups to build a topological space while in computationa topology persistent homology introduce the concept of filtration of simplicial complexes [3].
Recently, TDA has been applied in several studies of biological systems, for example Nicolau et al. [9], have defined a method that extracts information from high-throughput microarray data, the use of topology derive a more qualitative information than current analytic techniques. This identified a unique subgroup of Estrogen Receptor-positive (ER^+) breast cancers that express high levels of c-MYB and low levels of innate inflammatory genes. In [4], Chan et al., have applied topological methods to extend the limits of the phylogenetic tree for understanding the viral evolution.
Weight clique rank homology is a recent TDA technique, proposed by [11], meant to study complex networks, that allows to recover complete and accurate long-range information from noisy redundant network, by building on persistent homology. This first implementation of this technique has been proposed in *Holes*. In this work we implemented this new technique in a Java software suite named *jHoles* and available at [12]. Its core is a Java API efficiently implementing some algorithms to compute weighted clique rank homology. *jHoles* is the natural evolution of *Holes*, a Python package developed by Giovanni Petri within the EU funded project "Topdrim" [10]. *jHoles* is based on javaPlex [13,15,5]. *jHoles* fills the lack of an efficient implementation of the filtering process for clique weight rank homology. The main purpose of this paper is to give an overview of *jHoles* features, starting with a short introduction to the *Clique Weight Rank Persistent Homology* problem, discussing the algorithms and its implementation. We conclude with a biological case study showing how change the local connectivity of epidermal cells in response to a tumor presence. In appendix are given some useful definitions for graph theory, algebraic and computational topology.

2 Clique Weight Rank Persistent Homology

Clique weight rank persistent homology, sometimes referred as clique weight rank homology (CWRH or CWRPH for short), is a recent development in TDA, providing a new approach to the study of weighted networks. One of the main advantages of this approach is that it preserves the complete topological

and weight information, allowing us to focus on special mesoscopic structures: *weighted network holes*, that connect the network's weight-degree structure to its homological backbone [11].

2.1 Holes Algorithm

Holes is the first implementation of the clique weight rank persistent homology. The algorithm is based on the construction of a filtered simplicial complex, starting from all the maximal cliques of a network. The algorithm is structured as it follows:

- By varying the discrete parameter t scanning the sorted list of weight w_t of edges, build the subgraph with only edge-weights bigger than w_t. The sorting order is not relevant, it could be either descending or ascending; in this paper we will always refer to the standard descending CWRH.
- For each subgraph, find its maximal cliques.
- For each clique found, mark its rank t and its threshold w_t, then add it to the complex if not present yet (i.e. a clique should be added only the first time that it is found).
- For each element of the complex of size n, compute every combination from 0 to n of its elements to find missed faces.

Now we have a filtered simplicial complex that we may study with the preferred tool (for *Holes* it is javaPlex).
Even if this technique is really strong for weighted networks analysis, it has some open issues from a computational point of view. We realized by empirical measures that the 90% of *Holes* execution time is spent loading the graph (5-10%) and in the filtration process, while the rest (usually less than 10%) is spent calculating persistent homology. This is caused mostly by the clique finding problem; we recall that a clique is a complete subgraph of a graph (i.e. each node of the subgraph is connected with all the other nodes). The clique finding problem is a NP-Complete problem, and it is commonly assumed to run in $O(3^{\frac{n}{3}})$, since this is the maximum number of possible cliques in a graph [8]. At each index t, *Holes* recomputes the list of maximal cliques for the subgraph. Thus, it is recomputing the same thing for all the time, since maximal cliques of a graph contains every other possible clique [14]. Moreover, this approach depends both on the density of the graph and on the number of different weights assigned to edges: for large networks this becomes a problem.

2.2 jHoles

jHoles is the main outcome of this study. It is a Java API implementing the same functionalities of *Holes* but using efficient, fast algorithms. It is thought to be as fast as possible, it solves many of the problems related to memory

management and it adapts to the computer on which it is running (i.e. according the number of threads that are executed at the same time with the number of processors). *jHoles* is available at [12]. *jHoles* is written in Java as it is a powerful, flexible language that is widely used by the scientific community [2]. Moreover, Java is free and comes with a complete framework. As *jHoles* is developed in Java, it is compatible with every operating system that supports a JVM, but it requires Java 1.7. *jHoles* persistent homology engine is *javaPlex*, a Java library that offers all the needed methods to compute persistent homology (for more information refer to [13]). We choose javaPlex as it is one of the best software for computing persistent homology and, most important, it is far far better documented than the others. We choose JGraphT as data structure to handle graphs [1].

jHoles is designed to be easily used even by non computer scientists. Its main point of access is *jHoles*, a class offering all the methods to process a graph. This architectural choice was made to keep it simple to use, grouping in a single class its core functions. This interface comes with some pre-made, multithreaded parsers for files, supporting GEXF files, "edge list" files (sometimes referred as "sparse matrix representation" i.e. in Matlab) or a plain text file representing a matrix. It offers different methods to filter the network: one, marked as deprecated, uses *Holes* original algorithm; the others are various implementations of the improved algorithm: the difference is mainly in the optimization, e.g. how many threads the library should use, where to use caching technologies or to thread pooling to reduce the overhead. It is currently under active development a paged data structure to store the simplicial complex, as its dimensions may grow up easily: its aim is to avoid computational limits (i.e. the computer which it is running on has not enough RAM) at the price of some speed. The result of the filtration is stored in a *Hash Map* provided by the Java Runtime Environment.

jHoles maintains compatibility with Python, providing some methods to:

- Store the output of the computation maintaining compatibility with *Holes*.
- Serialize the simplicial complex in the form of a dictionary to be reloaded later with Pickle or cPickle.

It offers some basic analysis tools too, like some measurements of the graph (i.e. density, local density, average degree...) and some statistical analysis of the output (i.e. network hollowness).

jHoles main difference with *Holes* is its filtration algorithm. We may start from two observations about graphs and clique to improve *Holes* algorithm.

[2] The syntax for the *jar-file* execution is: *java -jar jHoles path_to_input_edge_list path_to_summarized_output path_to_extendend_output* - For technical issues about the execution, the JVM configuration, etc... please refer to jacopo.binchi@studenti.unicam.it

Let G be a graph and v a vertex:

1 For every maximal clique C of $G \setminus v$, either C continues to form a maximal clique in G, or $C \cup v$ forms a maximal clique in G. Therefore, G has at least as many maximal cliques as $G \setminus v$ does.

2 Each maximal clique in G that does not contain v is a maximal clique in $G \setminus v$, and each maximal clique in G that does contain v can be formed from a maximal clique C in $G \setminus v$ by adding v and removing the non-neighbors of v from C.

For other details, see [14]. From this point of view, we can easily summarize these observations in two cases. Let C_t be a clique at the step t, then there exist two possibilities:

- C_t is a maximal clique for G_t;
- C_t is a subgraph of a maximal clique in G_t;

This is really important because it allows us to build a more efficient algorithm. *Holes* algorithm adopts a constructive approach, but it is slow and heavy in terms of computation and resources, so we want to propose a different approach. We now know that if we want all the cliques in a graph, then they are either a subset of a maximal clique of the graph or a maximal clique. Then we can run the Bron-Kerbosch algorithm for the entire graph and then find all the other cliques looping on each maximal clique. So, the first three steps are:

Algorithm 1 - *Initial steps*
Step 1: Extract the sorted list of weights of G.
Step 2: List all the maximal cliques in G.
Step 3: For each maximal clique found, find all its subcliques.

In order to build the complex, we now need to rank these cliques: we now loop on each found clique to rank it. We actually look at each edge to found the minimum weight, that will correspond to the $t - th$ step of the filter. We then assign a label containing the step and the minimum weight to the clique. So, we add another few steps to our new algorithm.

Algorithm 2 - *Final steps*
Step 4: For each found clique, extract the list of its edges.
Step 4.1: For each edge, find the minimum weight. This is the rank of the clique.
Step 4.2: Look at the sorted list of weights to find the index corresponding to the clique rank.
Step 5: Put a label to the clique containing the rank and the weight.

This approach is really much faster than the previous one and it requires significantly less resources. Moreover, step 3 and steps from 4 to 5 can be executed in parallel. In fact, we can decompose each maximal clique independently from the others, and we can rank each clique (or set of cliques) separately from the others. A parallel implementation significantly improves performance on modern computers.

2.2.1 jHoles: network statistics

The last version of *jHoles* allows the user to calculate the most important network statistics that are the basis of most network analysis (e.g., for their comparison, classification, anomalies detection etc. . .) [7]. The following statistics are computed: *size, volume, average degree, maximum degree, clustering coefficient (local and average), negativity and networks weight.*

3 jHoles performance evaluation

Several tests for the performance evaluation of *jHoles* have been executed, in this section we will provide a short description of the datasets and of our results. For the sake of clarity, the platform that has been used for the computation is a middle-end desktop computer equipped with a quad-core processor and 4GB of RAM. It has a SSD hard drive to reduce loading time and the bottleneck usually represented by hard drives. Here the full configuration:

- CPU: Intel i5-2500k @4Ghz Turbo Boost disabled, Energy Saving disabled
- MB: AsRock Pro 3 Gen 3 z68 rev. B2
- RAM: 2x2GB DDR3 modules @1866mhz cl.7
- HDD: 128GB SSD
- OS: Windows 7 pro 64 bit Service Pack 1

Tests were executed with Java 1.7, python 2.7, jython 1.5 and mySQL 5.6

3.1 Datasets

3.1.1 US 2000 air passenger network

The network refers to the year 2000. The data used are publicly available from the website of the Bureau of Transportation Statistics[3]. Individual flights between airports were aggregated on routes as defined by origin and destination cities. The weight reported is the yearly aggregated passenger traffic.

[3] http://www.transtats.bts.gov/

3.1.2 C. Elegans neuron network

The network is a directed representation of the C. Elegans's neuronal network[4]. The network was symmetrized by summing the weights present on edges between the same nodes.

3.1.3 Twitter dataset

The dataset consists of a sample network of mentions and retweet between Twitter users and is available online on the Gephi dataset page[5]. Weights are proportional to the number of interactions between a pair of users.

3.1.4 Human gene

The gene interaction network used in the paper is a sampling of the complete human genome dataset available from the University of Florida Sparse Matrix Collection. Each node is an individual gene, while the edges correlates the expression level of a gene with that of the genes. The node set of the analyzed network was obtained by randomly choosing an origin node, then adding its neighborhood to the node set; the neighborhoods of the newly added nodes were then added to the node set recursively until a given number of nodes was obtained (in the case used the target number of nodes was N = 1300). Then all the edges present in the original network between the nodes in the node set were added, effectively taking a connected subgraph of the original network. To reduce the computational complexity due to the large density of the graph, the weighted clique fltration was stopped at an edge weight of 0.09.

3.1.5 Random generated datasets

We generated a few random graphs to show that *jHoles* performances are not influenced by the number of weights in the graph. We generated three graph, with density $\cong 1$, the first has $|W| = 10$, the second 1000 and the third 20000. This means that *Holes* algorithm needs respectively 10, 1000 and 20000 steps to stop, while *jHoles* has a constant run time.

3.2 Benchmark results

In this section we provide some graphs showing and studying benchmark results on these datasets. The first graph shows the time in seconds that *jHoles* and *Holes* took to process the datasets. We stopped the computation after 25 minutes if it was not complete.

[4] http://cdg.columbia.edu/cdg/datasets
[5] http://wiki.gephi.org/index.php/Datasets

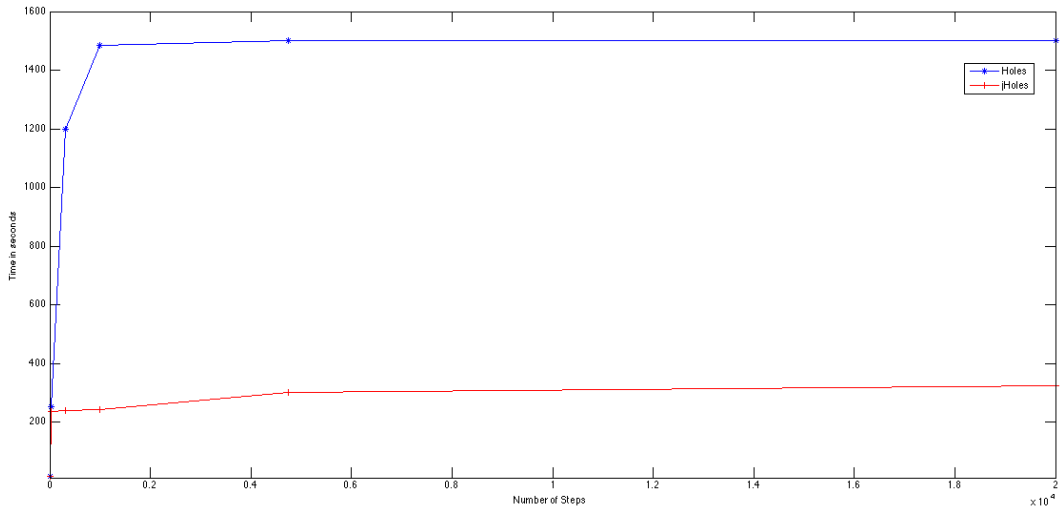

Fig. 1. Number of steps/time: *Holes* - blue line, *jHoles* - red line

The graph (see Fig. 1) represents the time (in seconds) in function of the number of steps of the graph. We can see that *jHoles* (red line) is not influenced by these parameters.

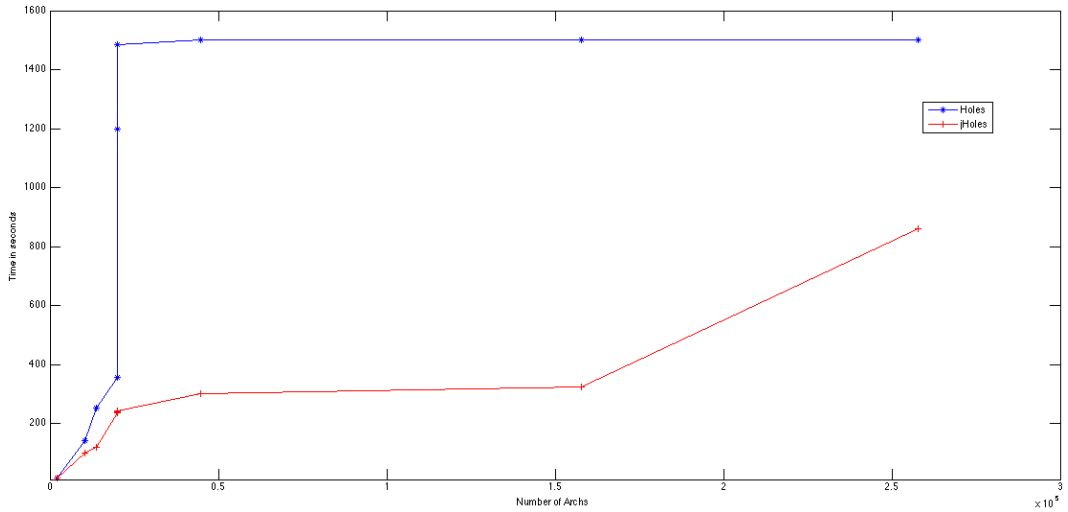

Fig. 2. Number of archs/time: *Holes* - blue line, *jHoles* - red line

The last graph (see Fig. 2) show how *jHoles* is influenced by the number of archs in the graph and how much it is more efficient than *Holes*. The last change of the slope in the red line (*jHoles*) corresponds to the paged dictionary activation.

In general we can conclude that all the following items have been needed

for the performance improvement:

- A new filtration algorithm.
- The optimizations of the general structure of the algorithm and its execution policy (e.g. the procedure selecting the right number of threads).
- Usage of properly data structure (e.g. Hash Map).
- Optimization of memory managment (e.g. paged dictionary).
- Full coding with Java.
- Right tuning of the JVM (e.g. heap size, garbage collector, etc. . .).

4 Biological case study: Analysis of epidermal cells before and after tumor

We applied *jHoles* for the study of the epidermis before and after a tumor [6]. The *in silico* model has been obtained following the indications provided in [6]. Briefly, models for tumor growth and skin turnover are combined with pharmacokinetic (PK) and pharmacodynamic (PD) models to assess the impact of two alternative dosing regimens on efficacy and safety. We studied the evolution of the topology (or the local connectivity). Epidermal cells sequentially pass three compartments, named proliferative (pc), differentiated (dc), and stratum corneum (sc) compartments. We obtained a network representation of the compartments connecting the cells using both their ammisible evolution (i.e., proliferative are connected only with differentiated and differentiated with stratum) and their concentration. The statistics of the networks are:

Before the tumor (see Fig.3):	After the tumor: (see Fig.4)
Number of Nodes: 98	Number of Nodes: 48
Number of Links: 393	Number of Links: 269
Average Node Degree: 4.01	Average Node Degree: 5.60
Average Clustering Coefficient: 0.190	Average Clustering Coefficient: 0.165

The homological analysis of the network for the healthy epidermis shows a higher number of holes that means a more spread cells distribution (the Betti numbers sequence: $\beta_0 = 1$ and $\beta_1 = 28698$), due to the presence of the three compartments. After the tumor the topology of network changed and the new

[6] The dataset is available at http://cuda.unicam.it/jHoles/dataset/

sequence of Betti number is $\beta_0 = 1$ and $\beta_1 = 24698$ with a reduced number of holes that means healthy cells disappeared and the network is less connected.

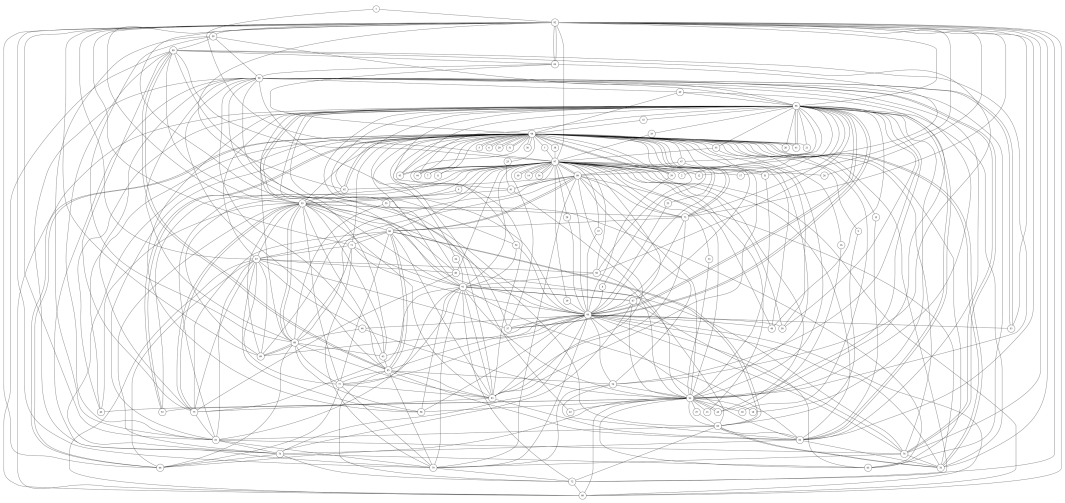

Fig. 3. Network of healthy epidermis

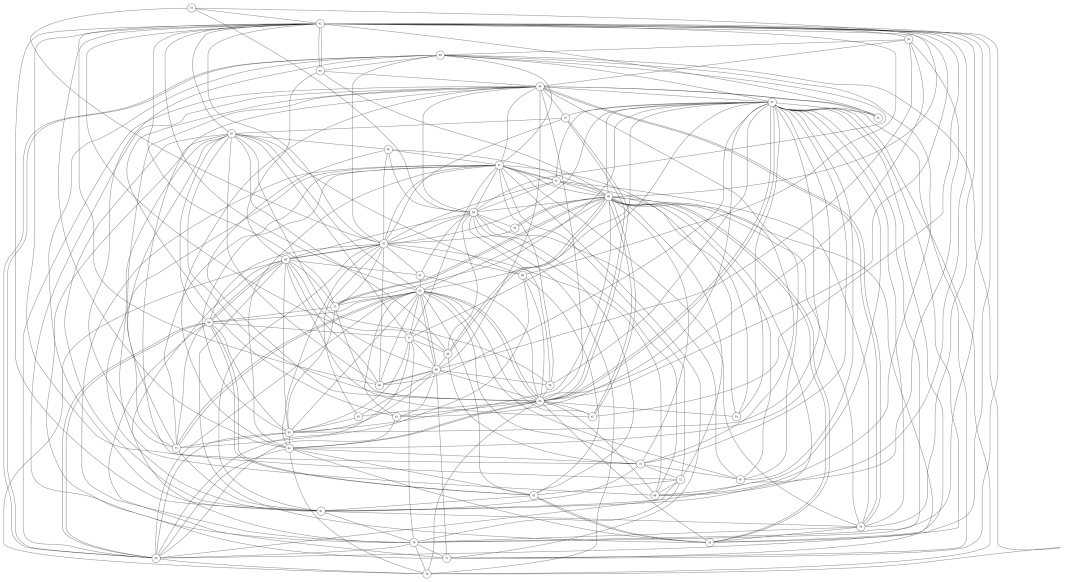

Fig. 4. Network of pathological epidermis

4.1 Conclusion and remarks

Currently we are developing a system that will recognize automatically both the generators of the holes and the persistent hubs, which will therefore allow both to recognize which type of cells were alive after the tumor. We believe that topological data analysis can be a useful instrument to study the drug effect.

Acknowledgements

We acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme (FP7) for Research of the European Commission, under the FP7 FET-Proactive Call 8 - DyMCS, Grant Agreement TOPDRIM, number FP7-ICT-318121.

References

[1] *Jgrapht, free java graph library*, Software available at http://www.jgrapht.org. URL http://www.jgrapht.org

[2] Bondy, J. A. and U. S. R. Murty, "Graph theory with applications," 6, MacMillan London, 1976.

[3] Carlsson, G., *Topology and data*, Bulletin of the American Mathematical Society **46** (2009), pp. 255–308.

[4] Chan, J. M., G. Carlsson and R. Rabadan, *Topology of viral evolution*, Proceedings of the National Academy of Sciences **110** (2013), pp. 18566–18571.

[5] De Silva, V., D. Morozov and M. Vejdemo-Johansson, *Dualities in persistent (co) homology*, Inverse Problems **27** (2011), p. 124003.

[6] Gieschke, R. and D. Serafin, "Development of Innovative Drugs via Modeling with MATLAB," Springer, 2013.

[7] Kunegis, J., *Konect: the koblenz network collection*, in: *Proceedings of the 22nd international conference on World Wide Web companion*, International World Wide Web Conferences Steering Committee, 2013, pp. 1343–1350.

[8] Moon, J. W. and L. Moser, *On cliques in graphs*, Israel journal of Mathematics **3** (1965), pp. 23–28.

[9] Nicolau, M., A. J. Levine and G. Carlsson, *Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival*, Proceedings of the National Academy of Sciences **108** (2011), pp. 7265–7270.

[10] Petri, G., *Holes: python package for persistent homology calculations*, "Software available at http://lordgrillo.github.com/Holes/" (2013).

[11] Petri, G., M. Scolamiero, I. Donato and F. Vaccarino, *Topological strata of weighted complex networks*, PloS one **8** (2013), p. e66506.

[12] Rucco, M. and J. Binchi, *jholes: A java high performance package for persistent homology*, "Software available at http://cuda.unicam.it/jHoles/ (2014).

[13] Tausz, A., M. Vejdemo-Johansson and H. Adams, *Javaplex: A research software package for persistent (co)homology*, "Software available at http://code.google.com/javaplex" (2011).

[14] Tsukiyama, S., M. Ide, H. Ariyoshi and I. Shirakawa, *A new algorithm for generating all the maximal independent sets*, SIAM Journal on Computing **6** (1977), pp. 505–517.

[15] Zomorodian, A. and G. Carlsson, *Computing persistent homology*, Discrete & Computational Geometry **33** (2005), pp. 249–274.

[16] Zomorodian, A. J., "Topology for computing," 16, Cambridge University Press, 2005.

Appendix: Useful mathematics definitions

The present work represents an interdisciplinary application from algebraic topology to graph theory, and an extensive treatment of these topics is not the main purpose of this paper. Moreover, to make the presentation in this paper self-contained, in the next subsection we will provide some mathematical definitions that can be useful to the reader. For a complete treatment of graph theory, algebraic and computational topology we suggest [2,16].

Graph theory

Definition 1 *Graph*
A Graph is an ordered pair (V,E) where V is the non-empty, finite set of its elements (nodes or vertices) and E is the non-empty finite set of its edges (links, ties or arcs), which are 2-elements subset of V.

Definition 2 *Weighted Graph*
A weighted graph is an ordered tuple (V,E,W, f), where V is the non-empty, finite set of its elements, E is the non-empty, finite set of its edges, W is the finite set of weights such that $|W| \geq 1$ *and f is a discrete function from E to W such that it associates each* $\epsilon \in E$ *to one* $w \in W$*.*

Definition 3 *Density of a graph*
Let G(V,E). Density of G, d(G), is defined as:

$$d = \frac{2|E|}{|V|\,(|V|-1)} \tag{1}$$

the ratio between the size of the graph and its maximum number of edges, so $0 \leq d(G) \leq 1$*.*

Definition 4 *Clique (complete graph)*
Let G(V,E) be an undirected graph (eventually weighted). G is a clique (or complete graph) if $d(G)=1$*.*

Definition 5 *Maximum weight clique (for weighted graphs)*
Let G and C be two weighted undirected graphs, with $C \subseteq G$ *and C clique, and for every edge of C, its weight is bigger (weaker) or equal a certain p. C is a maximum clique for the weight p if there is no vertex in G with weight bigger (weaker) or equal p that can extend C and for each clique* S_i *of G,* $|S_i| \leq |C|$*.*

Algebraic and Computational Topology

Definition 6 *Topology*
A topology on a set X is a family $T \subseteq 2^X$ such that

- *If $S_1, S_2 \in T$, then $S_1 \cap S_2 \in T$ (equivalent to: If $S_1, S_2, \ldots, S_n \in T$ then $\cap_{i=1}^{n} S_i \in T$)*
- *If $\{S_j | j \in J\} \subseteq T$, then $\cup_{j \in J} S_j \in T$.*
- *$\emptyset, X \in T$.*

Definition 7 *Topological spaces*
The pair (X,T) of a set X and a topology T is a topological space. We will often use the notation $\mathbb{X}$ for a topological space X, with T being understood.

Definition 8 *Simplices*
Let $u_0, u_1, ..., u_k$ be points in $\mathbb{R}^d$. A point $x = \sum_{i=0}^{k} \lambda_i u_i$ is an affine combination of the u_i if the λ_i sum to 1. The affine hull is the set of affine combinations. It is a k-plane if the k+1 points are affinely independent by which we mean that any two affine combinations, $x = \sum_{i=0}^{k} \lambda_i u_i$ and $y = \sum_{i=0}^{k} \mu_i u_i$ are the same iff $\lambda_i = \mu_i$ for all i. The k+1 points are affinely independent iff the k vectors $u_i \ldots u_0$, for $1 \leq i \leq k$, are linearly independent. In $\mathbb{R}^d$ we can have at most d linearly independent vectors and therefore at most d+1 affinely independent points.
k-simplex is the convex hull of k+1 affinely independent points, $\sigma = \{u_0, u_1, u_2, ...u_k\}$. We sometimes say the u_i span σ. Its dimension is $dim\sigma = k$. Any subset of affinely independent points is again independent and therefore also defines a simplex of lower dimension. The special names of the first few dimensions are:

- *vertex for 0-simplex;*
- *edge for 1-simplex;*
- *triangle for 2-simplex;*
- *tetrahedron for 3-simplex;*

Definition 9 *Face*
A face of σ is the convex hull of a non-empty subset of the u_i and it is proper if the subset is not the entire set. We sometimes write $\tau \leq \sigma$ if τ is a face and $\tau < \sigma$ if it is a proper face of σ. Since a set of k+1 has 2^{k+1} subsets, including empty set, σ has $2^{k+1} - 1$ faces, all of which are proper except for σ itself. The boundary of σ, denoted as $\mathrm{bd}\sigma$, is the union of all proper faces, and the interior is everything else, int $\sigma = \sigma -$ bd σ

Definition 10 *Simplicial complexes*
A simplical complex is a finite collection of simplices K such that $\sigma \in K$ and $\tau \in K$, and $\sigma, \sigma_0 \in K$ implies $\sigma \cap \sigma_0$ is either empty or a face of both.

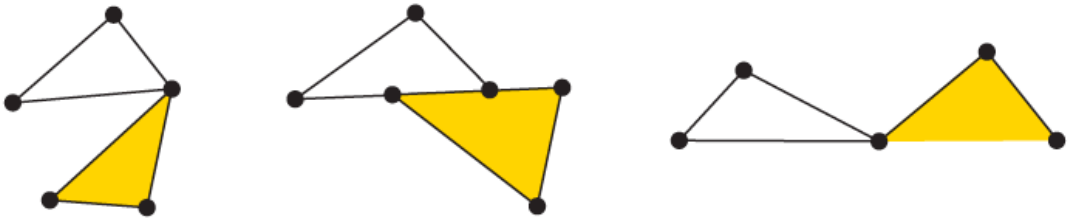

Fig. 5. A simplicial complex (left) and a collection of simplices (middle and right) which do not comprise a sumplicial complex.

Definition 11 *Filtration*
A filtration of a complex K is a nested sequence of subcomplex, $\emptyset = K^0 \subseteq K^1 \subseteq K^2 \subseteq .... \subseteq K^m = K$. We call a complex K with a filtration a filtered complex.

Definition 12 *Chain group*
The k-th chain group of a simplicial complex K is $\langle C_k(K), + \rangle$, let $\mathbb{F}$ be a field. The $\mathbb{F}-$linear space on the oriented k-simplices, where $[\sigma] = -[\tau]$ if $\sigma = \tau$ and σ and τ have different orientations. An element of $C_k(K)$ is a k-chain, $\sum_q n_q[\sigma_q], n_q \in \mathbb{Z}, \sigma_q \in K$.

Definition 13 *Boundary homomorphism*
Let K be a simplicial complex and $\sigma \in K, \sigma = [v_0, v_1, ..., v_k]$ The boundary homomorphism $\partial_k : C_k(K) \to C_{k-1}(K)$ is

$$\partial_k \sigma = \sum_i (-1)^i [v_0, v_1, ..., \widehat{v_i}, ..., v_n]$$

where $\widehat{v_i}$ indicates that v_i is deleted from the sequence.

Definition 14 *Cycle and boundary*
The k-th cycle group is $Z_k = ker\partial_k$. A chain that is an element of Z_k is a k-cycle. The k-th boundary group is $B_k = im\partial_{k+1}$. A chain that is an element of B_k is a k-boundary. We also call boundaries bounding cycles and cycles not in B_k nonbounding cycles.

Definition 15 *Homology group*
The k-th homology group is

$$H_k = Z_k / B_k = ker\partial_k / im\partial_{k+1}$$

If $z_1 = z_2 + B_k, z_1, z_2 \in Z_k$, we say z_1 and z_2 are homologous and denote it with $z_1 \sim z_2$

Definition 16 *k-th Betti number*
The k-th Betti number $\mathrm{B_k}$ *of a simplicial complex K is the dimension of the k-th homology group of K. Informally, β_0 is the number of connected components, β_1 is the number of two-dimensional holes or "handles" and β_2 is the number of three-dimensional holes or "voids" etc....*

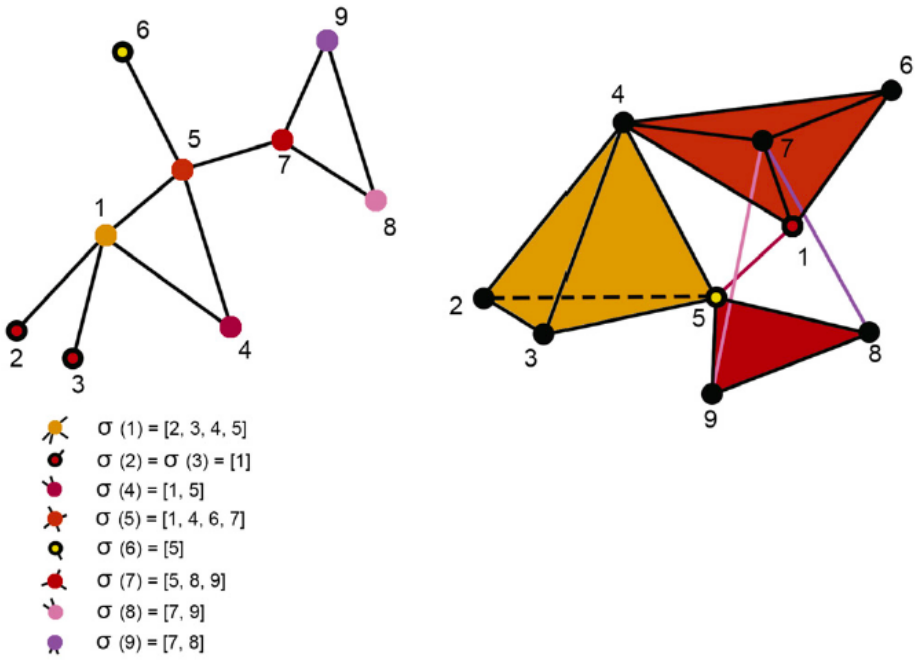

Fig. 6. From graph to simplicial complex with the expression for each simplex.

Definition 17 *Invariant*
A topological invariant is a property of a topological space which is invariant under homeomorphisms. Betti numbers are topological invariants.