FeatureCloud

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826078.

Privacy preserving federated machine learning and blockchaining for reduced cyber risks in a world of distributed healthcare

Project coordinator:
Professor Dr Jan Baumbach, TECHNISCHE UNIVERSITAET MUENCHEN (TUM)
info@featurecloud.eu

Deliverable D4.2
"Test report on different graph types"

Work package WP4
"Supervised Federated Machine Learning"

Disclaimer

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078. Any dissemination of results reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.

Copyright message

Document information

Grant Agreement Number 826078			**Acronym FeatureCloud**	
Full title	Privacy preserving federated machine learning and blockchaining for reduced cyber risks in a world of distributed healthcare			
Topic	Toolkit for assessing and reducing cyber risks in hospitals and care centres to protect privacy/data/infrastructures			
Funding scheme	RIA - Research and Innovation action			
Start Date	1 January 2019	**Duration**	60 months	
Project URL	https://featurecloud.eu/			
EU Project Officer	Reza RAZAVI (CNECT/H/03)			
Project Coordinator	Jan Baumbach, TECHNISCHE UNIVERSITAET MUENCHEN (TUM)			
Deliverable	D4.2 “Test report on different graph types”			
Work Package	WP4 “Supervised Federated Machine Learning”			
Date of Delivery	**Contractual**	M12 (31/12/19)	**Actual**	M22 (02/10/20)
Nature	REPORT	**Dissemination Level**	PUBLIC	
Lead Beneficiary	03 MUG			
Responsible Author(s)	Andreas Holzinger			
Keywords	Graph-based machine learning			

Table of Content

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

Acronyms and definitions

AI	artificial intelligence
API	application programming interface
BA	Barabasi-Albert
BFS	breadth-first search
concentris	concentris research management GmbH (Germany)
CNN	Convolutional Neural Network
CPU	Central processing unit
D	degree matrix
DB	database
FB	Facebook
FL	federated learning
GB	gigabyte
GND	Gnome Design SRL (Romania)
GNN	Graph neural network
GPU	Graphics processing unit
IID	Independent and Identically Distributed
KB	kilobyte
L	Laplacian matrix
MB	megabyte
ML	Machine learning
MUG	Medizinische Universität Graz (Austria)
NLP	Natural language processing
NN	Neural Networks
NN-based	Neural Network-based
PB	petabyte
RI	Research Institute AG & Co. KG (Austria)
SBA	SBA Research gemeinnützige gGmbH (Austria)
SDU	Syddansk Universitet (Denmark)
T	tree
TUM	Technische Universität München (Germany)
UM	Universiteit Maastricht (The Netherlands)
UMR	Philipps Universität Marburg (Germany)
vs.	versus
w.r.t.	with respect to
WWW	world wide web

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

1 Objectives of the deliverables based on the Description of Action

WP4 will help to contribute towards two very innovative and promising research streams: federated interactive machine learning, fostering client-side learning with the human in the loop; and explainable AI. The latter shall foster transparency and trust to the approaches developed in the FeatureCloud project. A huge challenge for federated machine learning is in graph parallelism. A solid understanding is important for the FeatureCloud project as there is a huge variety of different parallelization strategies (e.g., task, function, loop, event, data-structure).

Objective 2: To foresee the overall topology of a graph that was never seen in its entirety but only implicitly present via its distributed subgraphs and to experiment whether and to what extent such a graph can be thought of being connected in the first place; a subgoal is the exploration of link prediction via node similarity or feature-feature interaction as a necessary pre-processing step (Task 2).

Task 2: Test graph types and their suitability for federated learning approaches, particularly with bringing a human into the algorithmic loop (MUG, SDU). Specific graph types may significantly influence the choice of parallelization strategy, especially regarding communication requirements - a highly clustered, low-connectivity graph will be easier to partition via min-cuts than a randomly structured graph. An important question in this task pertains to the possibility of foreseeing the overall topology of a graph that was never seen in its entirety but only implicitly presented via its distributed subgraphs - and if such graphs can be thought of as being connected in the first place; otherwise some form of link prediction via node similarity or feature-feature interaction is a necessary pre-processing step.

Deliverable 4.2 Test report on different graph types

2 Executive Summary / Abstract

As the era of federated machine learning is upon us and edge computing gains momentum, the ML community is rapidly rethinking & reversing their old dogmas of big data and centralized computations, which require huge streams of data to be sent to algorithms that can be described in kilobytes. At the same time, Neural network architectures on graphs are a hot topic in the research community. In the context of our ongoing effort to establish the knowledge enabling future generations of client-centered graph-based machine learning (ML), we present this report on relevant graph types as well as design criteria for establishing graph libraries in software, two areas indispensable for the creation of effective data platforms.

Methodology

Although graphs provide us with a universal mathematical framework for many occasions, they are not automatically the best tool for every job, so we will evaluate their (dis)advantages w.r.t. medical image / ovarian cancer classification. The 4 most important aspects to examine involving graphs in our work will be their ability to 1) compete with traditional (NN-based) approaches, 2) produce explainable results, 3) facilitate distributed computation and 4) decrease exposure of sensitive data produced locally to the network of participants. In the following paragraphs, we give a short overview of interesting properties & research directions. Actual insights based on results can only be produced after extensive experimentation in a federated setting, which will be conducted in a later phase of the FeatureCloud project.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

Main results

We have deepened our understanding of graph types and come to the conclusion that rigid classifications - although interesting from a taxonomic or ontological point of view - are not very helpful in the application of graphs to a specific problem (not in the least because a specific problem domain will largely determine the graph-type for us). In general, we are interested in problems that can be modelled into groups of graphs with limited logical connections between them, since that will enable us to build distributed / federated ML applications without prohibitive communication overhead (the “gossip problem”). Moreover, keeping connection surfaces (groups of nodes that communicate across the network) to a minimum should also help in privacy aware learning.

Challenges

The major challenges in distributed graph-based machine learning will be the extraction of expressive & representative graphs from an underlying data-base, the choice and efficient implementation of a distributed computational paradigm, the automated extraction of explanations from the decision process as well as monitoring and minimizing the necessary communication / exchange of data between local subsystems.

Methodology

In extracting suitable graphs from medical images, we will proceed using a semi-automated approach involving image segmentation combined with a set of constraints provided by medical professionals; the goal is to produce robust graph representations while building a ‘graph-oriented’ vocabulary (how is a tumour cell defined as a node by its neighbourhood structure?) for a specific application domain (like ovarian cancer).
Building on that, we will formulate and implement a distributed classification algorithm based on the belief-propagation / message passing paradigm and compare its results to those of traditional (NN-based) methods. We will also closely monitor and examine different information propagation settings - from liberal to restrictive - and their effects on algorithmic performance.

3 Introduction and Motivation

Graph theory, albeit a discipline almost 300 years in the making and arguably reaching its high-point in the 1950s and 1960s during the cold war, has seen a revival over the past 20 years as social networks became ubiquitous, but also with the emergence of computational biology (protein-protein interaction networks), computational chemistry (prediction of potentially effective combinations of agents from exponential possibility spaces) as well as the need to provide quasi-real time navigation & logistics support in networks of millions of nodes (and upwards of hundreds of millions of edges).

Even more interestingly, graphs have been (re-)discovered by the Machine Learning community as Convolutional Neural Networks (CNNs) built for fast, non-linear pattern recognition in images can be potentially applied to graph structures as well. One of the hottest topics at the time of this writing is the discipline of *Federated Learning*, where a multitude of distributed nodes learn a global ML model together, exchanging only the minimum amount of information necessary to convey updates from their locally learned models.

Beyond federated learning (FL) lies edge computing, which reverses the flow of information back towards client devices, sending algorithms (whose source code can be measured in KBs and MBs) to where the data is generated and stored (measured in GBs or PBs) instead of the other way around; this approach can also deal much more efficiently (and liberally) with personal and even sensitive data, since they never leave the client machine. Considering that graphs are much more expressive

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

and human-understandable data structures than tables, lists or activation matrices, we can also hope for better explainability / transparency of algorithmic decisions, complying with legal requirements currently being (re-)formulated all over the world. Towards this end, we strive to explore the space of existing & potential graph architectures as well as graph classes / types that could help us bring about this new era of privacy-aware, distributed, explainable graph-based machine learning.

This report shall give a brief overview of available graph classes and some of their properties. Over the past 200 years hundreds, if not thousands of graph types have been explored, analysed and categorized. However, most of those are exhaustive enumerations of forms focusing on the number of vertices (k-regular, k-complete), positional arrangement (house graph, bucket ball) or possible derivations from other graphs (square of block etc.). Although this cornucopia of graph classes is fertile ground for mathematicians and graph theorists, we feel that our efforts towards building the foundation of a modern, graph-based federated learning architecture is best served by focusing on classes that are commonly encountered in practice, so the ensuing collection should be understood as our subjective (and in no ways nearly exhaustive) selection.

Suitability of graphs for Machine Learning - 4 aspects

1. Graphs and Neural Networks

As described in *D4.1 Survey on graph parallelism* many real-world problems can be formulated as graphs; however, unlike images or sequences of signals, graph structures do not exhibit rigidness (like pixel grids with fixed neighbourhoods) or regular context (like a sliding window over a time series). Therefore, it was a focus in recent years to formulate vectorized, highly parallel numerical computation strategies on them; this effort is still ongoing but has already yielded several interesting approaches which bring the power of GPU-based neural networking to bear on general networking problems. The question has therefore shifted from “*graphs vs. neural nets*” to “*graph modelling for neural nets*”; we foresee that a main future effort will lie in distributed NN-based graph computation, as we are also aiming towards this goal with our *local sphere learning* approach.

A major challenge in image-based ML is the fact that pixel-based representat-ions are very noisy, both w.r.t. artefacts introduced by image taking devices as well as superfluous information (e.g. tissue with no relevance to a classification outcome). In this sense, graphs can be understood as compressed information, crystallizing the relevant properties of an underlying real-world cloud of data points. This also means that multiple meaningful graphs can be extracted from the same underlying database (*views*) depending on the representation & learning objective, enabling us to examine which representations are most suitable to solving a problem. We will therefore compare results achieved by graph-based methods to those obtained via traditional CNNs, thereupon choosing to either deepen our efforts, keep graphs as auxiliary methods or dropping the methodology w.r.t. images.

2. Explainability

Since graphs are constructed of labelled entities & relations forming sequences of connections, decisions in a graph can be broken down into edge traversal rules, which are inherently explainable (we follow this link because the cell type is putatively the same, because a customer bought a product, etc.). This makes graph operations much more intuitive than activation signals in a neural network or k-dimensional embeddings. In short: edges are inherently semantic, whereas vectors or pixels are not. This also makes graphs suitable to construct so-called *counterfactual paths*, which describe a series of alterations to features of an input instance so that the predicted label of the instance would switch to a different (nearby) class. Constructing whole counterfactual graphs for a trained ML model would constitute a form of explainability, with the added benefit of being able to extract decision trees out of these graphs, thereby making the explanation easily understandable.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

3. Distributed / federated learning

There are many ways to use graphs for ML, but perhaps the one most suitable to distributed learning is belief propagation, which itself builds on the message passing paradigm. This technique has long (in the 2000s) revolutionized email spam filters by using the idea of similarity by similar environments (or, as known in NLP, “you shall know a word by the company it keeps” - Firth, 1957). The message-passing approach has been realized software-wise via the so-called Actor model, which is usually implemented via lightweight threads & message queues. Systems of this design have proven to be extraordinarily scalable w.r.t. data as well as number of computational nodes, and a critical question for us to explore will be its suitability for distributed computation over unspecialized networks (like the internet). Graphs are ideal data-structures for actor frameworks to operate on, since an actor could represent any entity on the granularity scale of a graph - a single node, a cluster, a component, or an entire (sub)graph - depending on the problem formulation.

This will enable us to detach graph-based computation from the localized level by computing embeddings for nodes, then on each level coarsening the graph structure into hierarchical clusters, which can be connected via message-passing to other graph clusters computed on spatially separated locations. Graph-based formulations of data are therefore natural candidates for distributed (thus federated) learning approaches.

4. Privacy-aware learning

Traditionally, privacy preservation is usually attempted via some form of *signal perturbation, differential privacy, or k-anonymization*. This however assumes that local data need to be exchanged to a certain degree, namely in a form carrying sufficient information for the receiving entity to be able to utilize it. With distributed machine learning, data are first and foremost used to generate local models, whose parameters are then *smoothed* over the network to arrive at a globally agreed-upon model. On a technical level, the message-passing paradigm introduced above, combined with graph representation learning, opens up new possibilities of achieving privacy by simply leaving out sensitive data when aggregating features for local embeddings that are subsequently propagated throughout the network.

Regarding medical image processing, classifying cancer by having nodes (representing cells) negotiate their latent classes amongst one another could enable us to learn on distributed batches of cancer graphs without having to reveal each local dataset to all participating systems in its entirety. Real-world experience / insights in this field are practically non-existent, which will allow us to contribute valuable heuristics.

4 Graph classes

4.1 Complete graph

A graph is said to be complete if every vertex v ε V has connections to all other vertices in the graph. Complete graphs of n vertices are denoted kn.
Due to the fact that in a complete graph each connected pair of vertices can be swapped without breaking the graph’s structure, there are *n*! automorphisms of a complete graph.

4.2 Complement graph

The complement graph (denoted G) is a graph G’ = (V, E’) which is formed by taking a source graph G = (V, E) and setting e ε E’ iff e !ε E and vice versa; meaning each existing edge in E is omitted in E’ and each non-existing edge in E is set in E’. The complement of a complete graph is therefore a

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

graph with an empty edge set E = {}. See Figure 1 for an example of a complement on the Petersen graph.

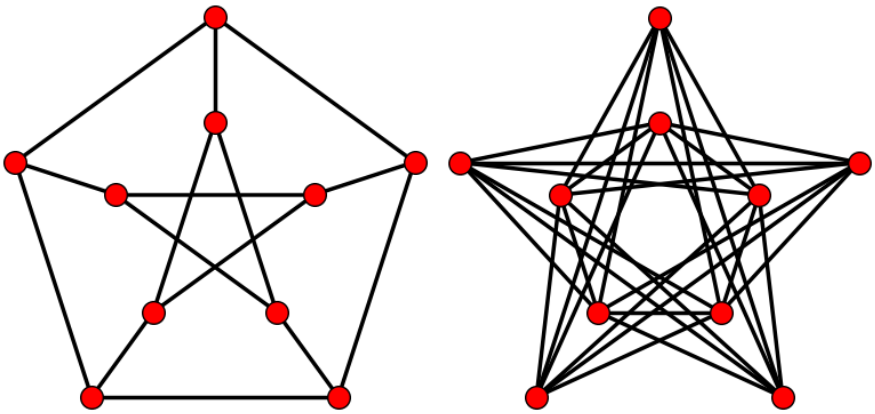


Figure 1: *Petersen graph and its complement. Source: Bernd Malle*

4.3 Directed vs. undirected

Edges in undirected graphs simply denote some kind of relation between the vertices they connect. In a traffic network such edges could represent roads, whereas vertices represent locations like street corners, intermediate waypoints etc. Obviously, routes between locations cannot always be passed in both directions, a fact represented as directed edges in graphs. Such directions can also occur in social networks (A ’loves’ B, but not vice-versa) or state-graphs (entropy).

Although direction seems trivial at first glance, it is a concept with important ramifications: e.g. the adjacency matrix of undirected graphs is symmetric which has consequences for *spectral* approaches; also there are algorithms which are only defined on directed graphs, such as *strongly connected components* or topological sorting. It is noteworthy that an undirected graph can always be converted to a directed graph by replacing each undirected edge with a pair of directed ones.

4.4 Weighted vs. unweighted

A graph is said to be weighted if edges between nodes are assigned some importance value. For instance, one can imagine a "friendship" edge as being simply a notation of people knowing each other, where an existing edge would symbolize a boolean value of 1. In a weighted scenario, we could additionally attribute some intensity to this friendship relation, so that a sense of relative closeness of actors arises. Likewise, in a state graph with vertices depicting weather conditions and edges the transition from one condition to another (’sunny’ → ’rainy’), we can easily see that not all transitions are equally likely (e.g. ’heat wave’ → ’blizzard’ would be very improbable).

4.5 Dense vs. sparse graphs

Imagine a social network with n people, where n is a large number (possibly in the millions or billions) and its adjacency matrix, where each row stands for the contacts of one person. As most people only have a very limited number of friends, most of the entries in each row will be zero (or undefined). As an example, Facebook (FB) in late 2019 boasted about 2.5 billion monthly active users. Assuming that an average user on FB has about 250 connections, each row in the adjacency matrix will only have 1 in 1e7 places set to 1, with all the other ones set to zero. One can easily see that storing the full adjacency matrix is therefore a futile undertaking, especially since algorithms executing against this data-structure would waste most of their CPU cycles computing null values.

Thus, an improved format for efficiently storing & computing sparse matrices was needed; although intuitive replacements like *linked lists* or *hash maps* help to reduce space requirements, they are too slow (distributed locations in memory) to be efficient on real-world networks. The Compressed

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

sparse row 2 format (see Figure 2) was introduced as an alternative: It encodes each non-zero (defined) value in the adjacency matrix as a triplet: the row-number, the column-number as well as the actual value. This can be further optimized by compressing row values with pointers; however, it will take on the order of 3 values for each non-zero value in the original matrix, rendering the compression useless for densities $d > 1/3$.

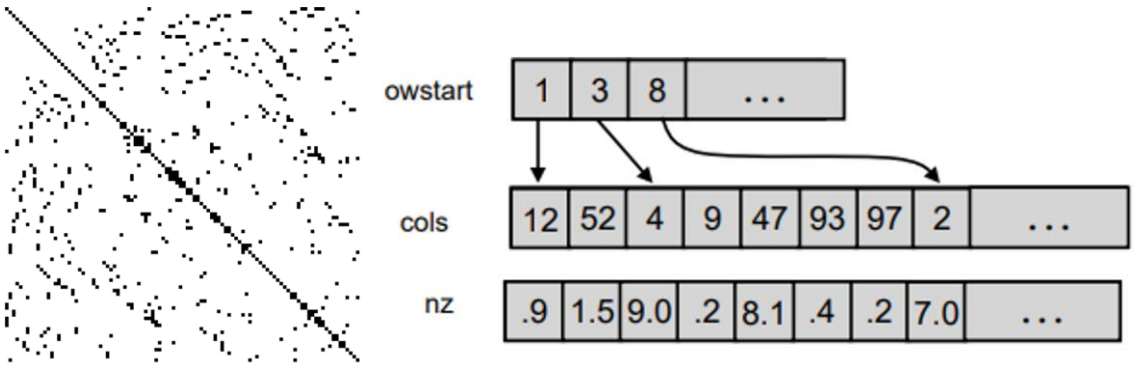


Figure 2: *Sparse matrix and Compressed Sparse Row (CSR) representation. Source: [1]*

4.6 Spectral graphs

Rather than being a class of graphs themselves, spectral graph theory as more an additional aspect of how graphs can be represented & viewed in general. Since every graph can be represented by matrices (the adjacency matrix, reach-ability matrix - also known as transitive closure, the degree matrix etc.), one way to approach graph-theoretic problems is by the properties of and operations applicable to those matrices. For instance, the general graph partitioning problem of putting nodes into groups such that the inner-to-outer density ratio is high, can be formulated as a sorting problem on the adjacency matrix, as Figure 3 shows.

In spectral graph theory, the *Laplacian matrix L* (also called *admittance* or *Kirchhoff's* matrix) is a construct of special attention, since it displays some traits useful for the calculation of common graph problems.

As an example, the second smallest eigenvalue of *L* approximates the sparsest cut of a graph, the number of connected components is the dimension of the *nullspace* of the Laplacian, and it can also be used to calculate the number of spanning trees for its graph. Furthermore, *L* can be used to construct low-dimensional graph embeddings, with use cases in machine learning as well as visualization;

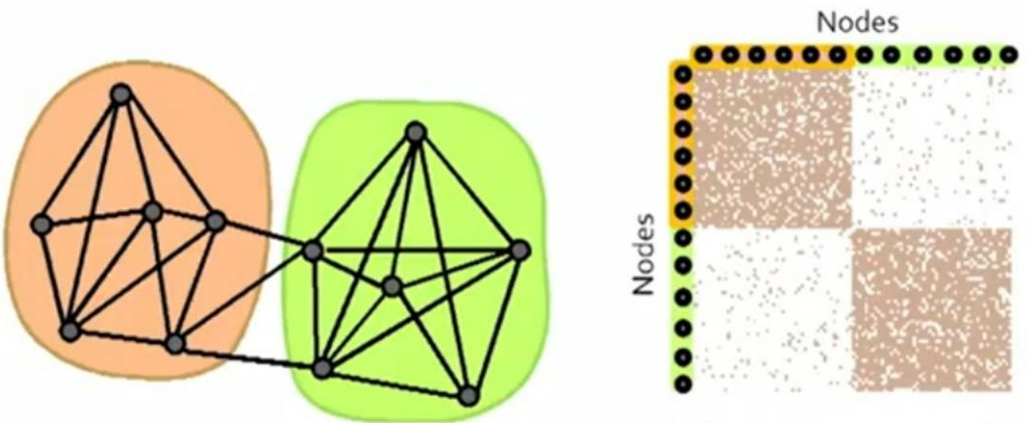


Figure 3: *Graph partitioning, here in the form of community detection, as a spectral clustering problem, i.e. an ordering on the graph adjacency matrix; the goal is to order vertices in such a way that the adjacency*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826078.

matrix becomes "dense" within clusters and "sparse" in between, resulting in characteristic block-like shapes along the matrix diagonal. Right the network; left the adjacency matrix. (*Source: Mining of Massive Datasets, Jure Leskovec, Anand Rajaraman and Jeffrey D. http:// www.mmds.org.)*

The Laplacian is computed as *L = D - A*, where *D* is the degree matrix (containing non-negative integers) and *A* is the adjacency matrix (containing only 0s and 1s with all 0s on the diagonal) of the graph. The elements of the Laplacian are therefore given by:

$$L_{i,j} := \begin{cases} deg(v_i) \text{ if } i = j \\ -1 \text{ if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 \text{ otherwise} \end{cases}$$

In summary, spectral graph theory studies the properties of graphs via the eigenvalues and eigenvectors of their associated graph matrices.

4.7 Trees

Since trees are widely used in computer science for a multitude of applications, they are often regarded as a data structure in their own right. Precisely speaking however, a tree T is a special class of graph in that T is a connected graph that has no cycles.

According to [2], given a graph *T* with *n* vertices, a tree *T* is connected and has no cycles, has *n*-1 edges, the removal of any edge disconnects *T* , any two vertices of *T* are connected by exactly one path, and the addition of any new edge creates a cycle.

The most commonly used form of trees are *binary trees*, e.g. in binary decision trees or for sorting problems (if only as a logical construct which is actually implemented as an array).

Rooted trees feature one distinguished node as a "starting point"; a *spanning tree* is a subgraph of a graph which contains all its vertices and is also a tree; a *forest* is an acyclic directed graph consisting of a disjoint union of trees; Cayley’s formula shows that for any positive integer *n*, the number of trees on *n* labelled vertices is n^{n-2}.

4.8 Isomorphic & Automorphic graphs

In order to define *-morphism we first have to define functions & mappings of sets:

1. A *mapping* is a projection of an element in one group *A* to an element of another group *B*.
2. A *general function* is a rule or a set of rules that maps every element of a group *A* to an element of a group *B*.
3. An *injective function* is one that maps every element of *B* exactly once (no element in *B* is mapped onto by more than one element in *A*).
4. A *surjective* function is a function such that every element in *B* is mapped (hit) at least once.
5. A *bijective* function is a combination of an injective and a surjective function, resulting in a perfect *one-to-one* relationship.

4.9 Isomorphism

A graph isomorphism (example see Figure 4) is a bijection preserving adjacency of the vertices in a graph. The problem has been studied for decades, and even back in 1970 Corneil & Gotlieb [3] pointed out that no efficient deterministic algorithm is known for determining whether two given finite graphs, tt1 and tt2, are isomorphic (although for many special classes of graphs, isomorphism can be solved in polynomial time).

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

Observing that even the most trivial isomorphism testing algorithms achieve good performance on random graphs, Babai [4] designed an algorithm which, for almost all graphs *X*, tests any graph *Y* for isomorphism to *X* within linear time. This work was further improved upon by Bruce [5] who focused on solving practical problems, Cordella [6] for application to large graphs as well as Babai [7] in extending their algorithm to String isomorphism and Coset Intersection achieving quasi-polynomial time.

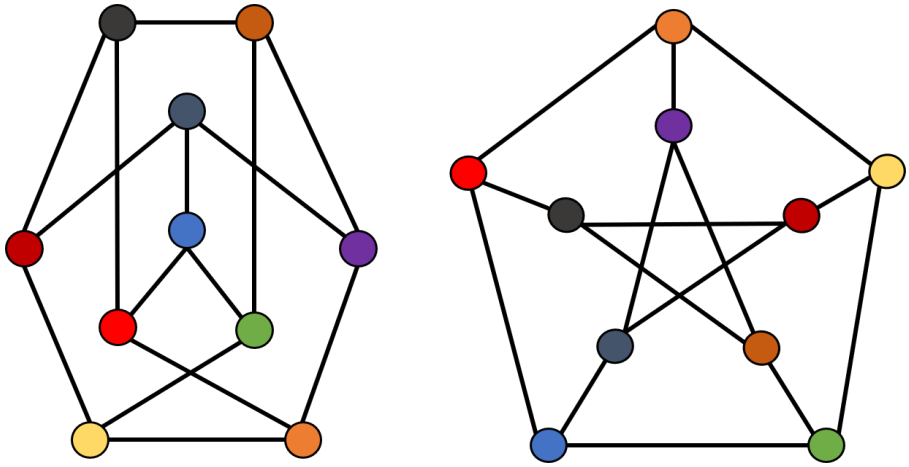


Figure 4: *Example of graph isomorphism on a graph of 10 vertices and 15 edges (Petersen graph). Source: Bernd Malle.*

4.10 Automorphism

An automorphism is an isomorphism from one graph to itself, and it often implies symmetries in a graph, such that applying a geometric transform - identity (fixing), rotation, reflection (mirroring), etc. (see Figure 5 for examples of mirroring & rotation) results in an "equivalent" graph structure with only positions or labels of vertices swapped / rearranged.

For instance, figure 5 shows three possible automorphisms, the first of which could be described as a function $\alpha : V(G) \rightarrow V(G)$ such that α maps $A \rightarrow A$, $B \rightarrow C$ and $C \rightarrow B$.

Automorphism have some interesting properties that should be mentioned:

1. The *identity map* (fixing of all nodes) is always an automorphism (*"base case"*, so each graph has at least one automorphism).
2. If α and β are automorphisms of G, then $\alpha \cdot \beta$ is also an automorphism of *G*.
3. If α is an automorphism of *G*, then α^{-1} is also an automorphism of *G*.
4. The more automorphisms a graph has, the more symmetric it is.
5. For every graph G, Aut(G) ~= Aut($\underline{G}$), that is the automorphism group of G is isomorphic to the automorphism group of the complement graph $\underline{G}$.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

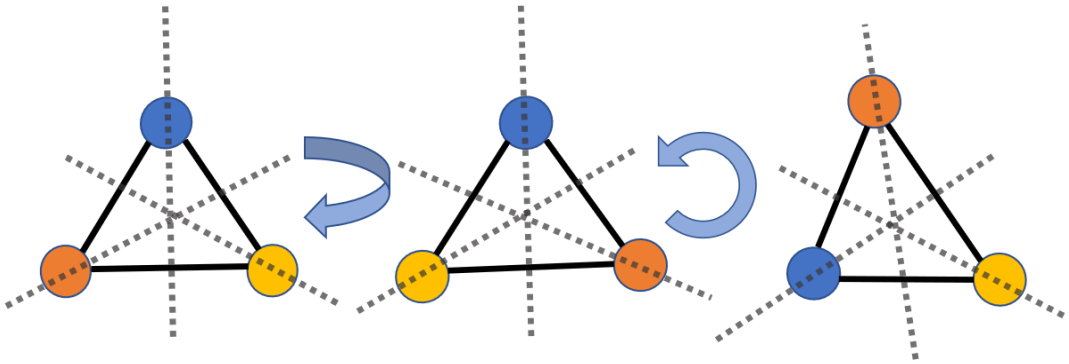


Figure 5: *Example of graph automorphism. By flipping the graph along one of the shown axis, then rotating, the positions of the vertices are altered while adjacencies & non-adjacencies are perfectly preserved. Source: Bernd Malle*

4.11 (Strongly) Regular graphs

A regular graph is a graph in which each vertex has the same number of direct neighbours, meaning the degree of each vertex d(vi)) is the same. In a directed graph, this condition must hold for both in-degrees as well as out-degrees. A regular graph with vertices of degree k is called a k-regular graph. Figure 6 depicts k-regular graphs for k ε {0, 1, 2, 3}

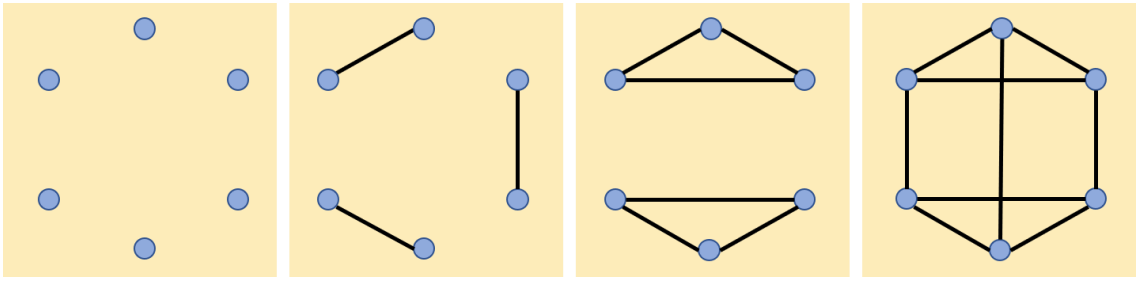


Figure 6: *Examples of regular graphs. Source: Bernd Malle*

Strongly regular graphs were introduced by Bose [9] in 1963; Meringer [10] introduced an efficient algorithm for the generation of strongly regular graphs with a given amount of vertices *n* and a desired degree *k*. The computational intensity to generate such graphs is illustrated by the fact that they boast the capacity of their implementation to generate the 5-regular graphs on 16 vertices for the very first time (and 5-regular graphs with girth 5 and minimal number of vertices were generated in less than one hour.)

Cameron [11] observes that strongly regular graphs lie somewhere between the highly structured and the apparently random. They define a strongly regular graph with parameters (n, k, λ, μ) as a regular graph on n vertices with degrees *k* and the following properties: 1. any two adjacent vertices have exactly λ common neighbours & 2. any two non-adjacent vertices have exactly μ common neighbours. The complement of a strongly regular graph is strongly regular itself.

4.12 Rigid / spatially local graphs

Although not an official class of graphs, we nevertheless deem it necessary to mention rigid or spatially local graphs, denoting graphs in which all vertices have the same local environment or neighbourhood structure. Such a structure can be pictured like pixels in an image, where for each (non- edge) pixel there exists a regular 8-neighborhood (at least in 2-dimensional pictures). This property is very useful in modern machine learning applications, since it is required for Convolutional Neural Networks (CNNs) to work.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

In the absence of *spatial locality*, some form of regularization must be applied to a graph in order to normalize it into feature vectors of equal length; the works of [12] (deepwalk) and Niepert, Ahmed & Kutzkov [13] are good examples and have already been mentioned in our *graph parallelism* report.

4.13 (k-)partite graphs

A graph is bi-partite if it consists of two sets of vertices V1, V2 and edges E such that no edge connects two vertices of the same group. For instance, in a co-authorship graph consisting of vertex sets V1=*authors* and V2=*papers* as well as an edge set E=*authored*, edges can only exist between V1 and V2, since no author authored another author and no paper authored another paper. A complete bi-partite graph is one in which every vi ε V1 is connected to all vj ε V2, meaning it has maximum degree without breaking the partition criterion.

Such notions can be generalized to *k*-partite graphs, in which the vertices of a graph fall into *k* independent sets, see figure 7. While bi-partite graphs can be recognized in polynomial time, for any $k > 2$ that problem is NP-complete. *k*-partitioning can also be understood in terms of vertex colouring, because any such graph can be coloured with *k* colours such that no adjacent vertices share the same colour.

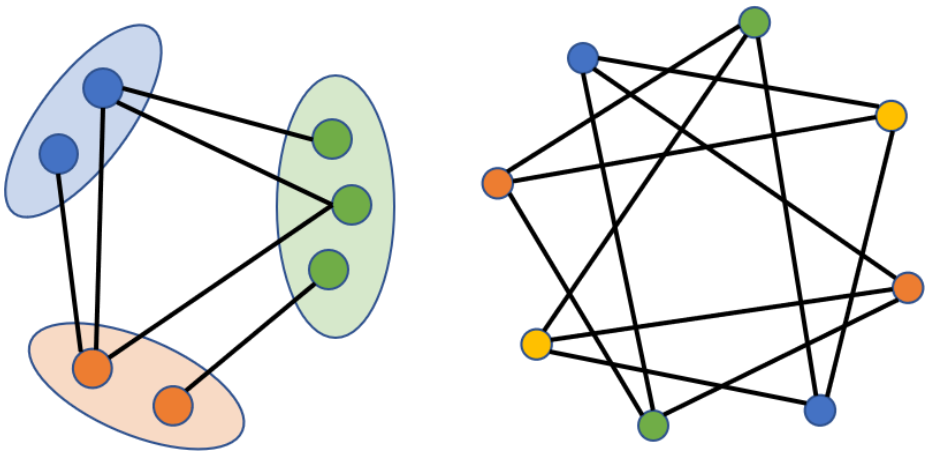


Figure 7: *Two examples for k-partite graphs (left: 3-partite, right: 4-partite). Source: Bernd Malle*

4.14 Scale-free graphs

In a scale-free network the degree distribution follows a power-law, rather than the Poisson distribution of the classical random graph models G(n, p) and G(n, m) [14], [15], [16]. This follows from an implicit "policy" in which new (or existing) vertices attach themselves to other vertices with a probability proportional to that target vertex’s degree - this is called the preferential attachment or the BA (Barabasi-Albert) model [17], see figure 8.

According to Cohen & Havlin [18], in contrast to the diameter of regular random networks or small world networks which is known to be $d \sim \ln(N)$, scale-free networks with $2 < \lambda < 3$ have a much smaller diameter, behaving as $d \sim \ln(\ln(N))$.

Applying an edge list partitioning technique, [19] explored techniques to process large scale-free graphs in distributed memory and demonstrated the effectiveness of their approach using Breadth-First Search (BFS), K-Core decomposition, and Triangle Counting on significantly larger datasets than comparable systems.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

Li et al. [20] observed that many approaches in the field of scale-free networks are not founded on precise, mathematical definitions, and therefore introduce a structural metric that allows to differentiate between all simple, connected graphs having an identical degree sequence.

4.15 Hypergraphs

A hyper-edge is a generalization of an edge connecting an arbitrary number of vertices of a graph (a set of vertices of arbitrary size); a hypergraph is a graph containing at least one hyper-edge.

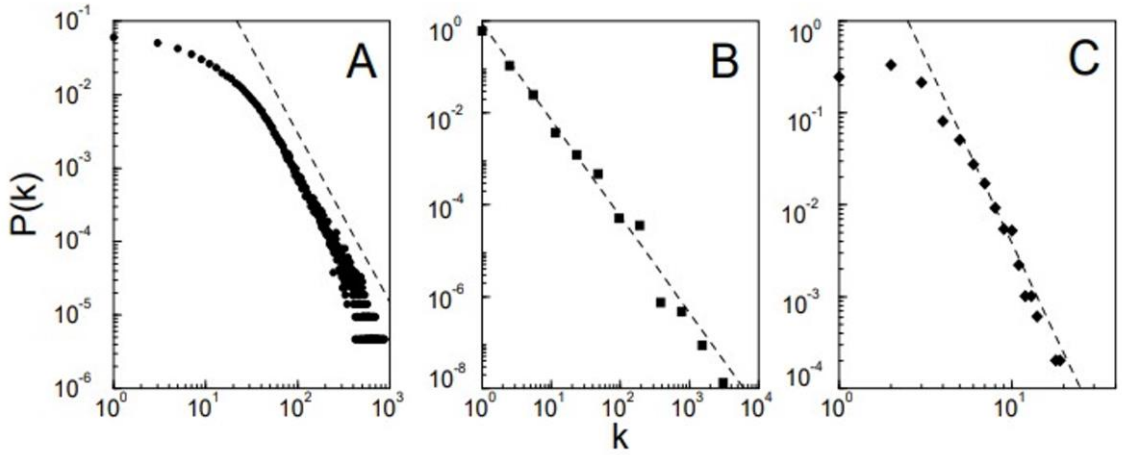
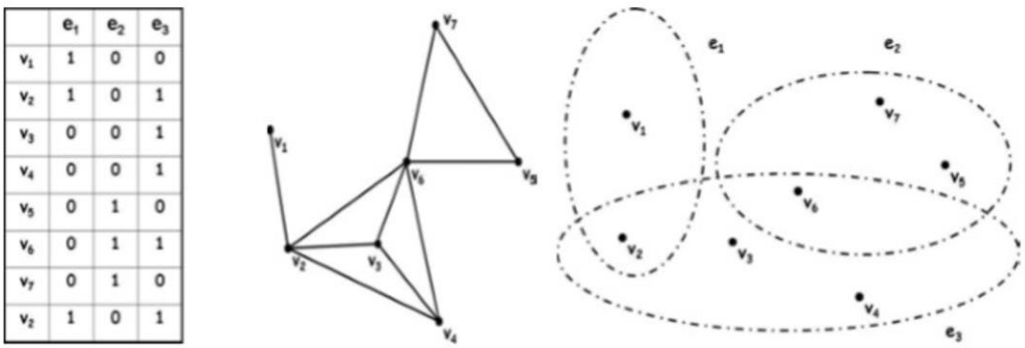


Figure 8: *Power law distribution functions of connectivities for selected large networks according to Barabasi-Albert [17]. (A) actor collaboration graph with N=212,250 vertices and average connectivity k = 28.78; (B) WWW, N = 325,729 with k=5.46; (C) Powergrid data N = 4,941 k=2.67; the dashed lines have slopes of A = 2,3, B = 2,1 and C = 4*

The set of hyper-edges *H* can also be seen as a subset of the power-set of vertices. Since vertices in a hypergraph have rank, hypergraphs can be k-regular. A hypergraph is normal *iff* the maximum number of disjoint hyper-edges coincides with the minimum number of vertices representing the hyper-edges in each partial hypergraph of it [21]. Just like regular graphs, hypergraphs can be represented and computed via methods of spectral graph theory [22].

The authors of [23] point out that traditional pairwise connections of objects are often insufficient formulations of real-world settings and demonstrate this with an example of a co-authorship network (figure 9). They then go on to propose hypergraph partitioning methods based on random walks and spectral graph theory.



	e_1	e_2	e_3
v_1	1	0	0
v_2	1	0	1
v_3	0	0	1
v_4	0	0	1
v_5	0	1	0
v_6	0	1	1
v_7	0	1	0
v_2	1	0	1

Figure 9: *Example of a hypergraph formulation of a co-authorship network. On the left we see an author set E={e1,e2,e3} and an article set V={v1,v2,v3,v4,v5,v6,v7}. The entry (vi,ej) is set to 1 if ej is an author of article vi, and 0 otherwise; in the middle we see an undirected graph in which two articles are joined together by an edge if there is at least one author in common. This graph cannot tell us whether the same person is*

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

the author of three or more articles or not! On the right side finally we see a hypergraph which completely illustrates the complex relationships among authors and articles. Source: [23].

Gallo et al. [24] concerned themselves with directed hypergraphs and introduce the notion of paths and hyperpaths; they explore cuts, cut-sets and minimum path problems and conclude with the insight that hypergraphs are powerful tools to solve problems in areas like propositional logic, databases and urban transportation.

4.16 Random graphs

Random graphs emerge when one randomly inserts edges into an initially empty edge set E = {}, that is picking edges from the set of all (n choose 2) possible edges with a certain probability [25]. Although there are many possibilities to sample the possibility space, two models are among the earliest and most popular generators:

The *random binomial* or *G*(*n, p*) graph model has two parameters, *n* for the number of vertices and a probability *p* with 0 ≤ *p* ≤ 1. The model assigns to a graph *G* the probability:

$$Pr[G] = p^{|E(G)|}(1-p)^{\binom{n}{2}-|E(G)|}$$

The *Uniform random graph* or *G*(*n, m*) model also has two parameters, *n* for the number of vertices and *m* for the number of edges with 0 ≤ *m* ≤ (n choose 2). A random graph is therefore sampled with probability:

$$Pr[G] = \begin{cases} \frac{1}{\binom{\binom{n}{2}}{m}} \text{ if } |E(G)| = m \\ 0 \text{ if } |E(G)| \neq m \end{cases}$$

We also want to mention *Kronecker graphs*, which in their original definition do not fall into the category of random graphs (although there is a stochastic version [26]) but are a famous sub-type of graph generators, in which a graph is *"expanded"* from a very small initial adjacency matrix by iteratively applying the Kronecker product, see figure 10.

$$\underset{N\,x\,M}{\mathbf{C}} = \underset{N\,x\,M}{\mathbf{A}} \otimes \underset{K\,x\,L}{\mathbf{B}} \doteq \underset{N*K\,x\,M*L}{\begin{pmatrix} a_{1,1}\mathbf{B} & a_{1,2}\mathbf{B} & \dots & a_{1,m}\mathbf{B} \\ a_{2,1}\mathbf{B} & a_{2,2}\mathbf{B} & \dots & a_{2,m}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1}\mathbf{B} & a_{n,2}\mathbf{B} & \dots & a_{n,m}\mathbf{B} \end{pmatrix}}$$

Figure 10: *Kronecker product of matrices A and B. Source: [27].*

A graph is *chordal* if every cycle of at least 4 nodes contains a chord, that is an edge connecting two vertices within a cycle that is not part of the cycle itself.

A *perfect* graph is a graph in which the chromatic number of every induced subgraph equals the size of the largest clique of that sub-graph (clique number). Perfect graphs are interesting in the sense that they allow polynomial-time solving of certain general graph problems (maximum-clique, coloring, etc.).

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

A graph is said to be *k-connected* if it cannot be disconnected by removing less than k vertices. For instance, a 1-connected graph can be split into separate graphs by removing only 1 (particular) vertex, whereas one can remove any arbitrary vertex from a 2-connected (or biconnected) graph without fracturing it. In the context of connectivity Menger’s Theorem states that the minimum size of a (*u, v*) cut set is equal to the maximum amount of disjoint paths from *u* to *v*.

A graph is said to be *planar* if it is possible to draw it on a 2D surface without any of its edges intersecting, except at their endpoints, see figure 11. Schnyder [28] showed that each plane graph of order *n*=3 has a straight line embedding on the *n*-2 by *n*-2 grid and that this embedding is computable in time *O*(*n*).

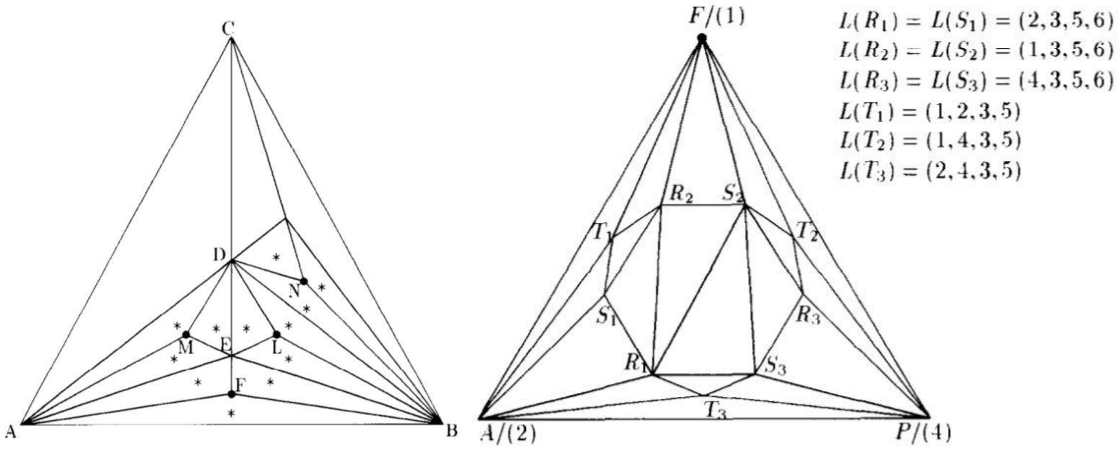


Figure 11: *Examples of planar graphs [8]*

4.17 How graph types were chosen for inclusion in this report

Faced with a cornucopia of available graph classes, we had to pre-select the most relevant ones for our work to keep this survey desirably brief. *Complete graphs* are the hardest type to partition, since every node has connections to every other node, and therefore offer the least potential for decentralization. *Complement graphs* could be interesting in peer-to-peer settings as they could represent identical data domains with complementary connection information (= link structure), so that signals passed between them would carry maximal information. *Directed* and *weighted* graphs are fundamental to every discussion about networks and could influence the way signals are propagated. The notion of a *spectral graph* conveys the assumption of global knowledge of a graph structure at processing time, which renders those methods unsuitable for distributed learning. *Trees* are strictly hierarchical structures that naturally enable coarsening & align perfectly with the Federated Learning methodology. *Isomorphisms* and *automorphisms* are used as similarity metrics in (sub)graph matching problems and thus represent crucial tools for building collaboration graphs across local spheres. *Strongly regular* graphs show uniform distributions of neighbourhood sizes which might be exploited as an assumption in belief propagation. *Rigid / spatially local* (=grid) graphs are natural candidates for processing via Convolutional nets since they do not need pre-processing to normalize their neighborhoods into fixed-length feature vectors. *K-partite* graphs exhibit regular traversal patterns (every n^{th} distant node belongs to a different cluster, every $(2n-1)^{th}$ node belongs to the same) which could impact the way of partitioning such graphs. *Scale-free networks* show the same kind of patterns on different levels of granularity (=scale), a property which lends itself to hierarchical feature learning. *Hypergraphs* often occur naturally (e.g. interactions between more than 2 molecules) and shift the concept of clusters which is central in graph partitioning. There is a rich literature on *random graphs* and the effects of their properties on partitioning, message passing, graph isomorphism testing etc.

As can be seen from the above, the effects of certain topologies on *clustering/partitioning, overall network layout, message passing* as well as *feature learning* with tools of modern graph

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

representation learning (especially those based on the message-passing paradigm) was the key criteria for inclusion of a particular graph type. However, we need to point out that there are many graph classification schemes, thus our selection remains a subjective one.

4.18 How graph classes pertain to challenges in Federated Learning

Generally, the type & topology of a graph will only influence the design of a distributed system under the assumption that the graph represents a global database which is *split* into different parts that are subsequently processed in a distributed / parallel manner. This report’s research subject as a whole was motivated by the assumption that such a scenario would hold true for the requirements of the FeatureCloud project. However, we came to the conclusion that graph topologies will be imposed on us by the reality of existing data bases within member institutions, so that we’ll have little control over their structure. Consequently, the following is a very brief discussion of possible links between network structure and Federated Learning:

1. *Expensive communication.* Theoretically, splitting dense graphs into subgraphs for the purpose of federated learning will yield large connection surfaces (= large sets of edges spanning individual subgraphs) which will result in increased network traffic. Likewise, graphs with high modularity & low connectivity would constitute ideal candidates for FL.
2. *Systems Heterogeneity* (in the sense of storage & processing capacity) is completely unrelated to graph topology.
3. *Statistical Heterogeneity*. Often, data generated in a medical setting are not naturally provided in the form of a graph, which entails the necessity to apply procedures like *node matching & link prediction* if one wished to construct a graph out of such data. In case input data are not IID, this could result in different graph topologies emerging across different members of a FL network. However, if and to what degree such aberrations would influence downstream ML tasks lies in the shape of the data itself, since they would logically necessitate the resulting graph topology.
4. *Privacy concerns*. Under the assumptions that all members in a federated setting exchange compressed, conceptualized representations of data (=*embeddings*) under a message-passing paradigm, the greatest privacy concerns lie in security (man-in-the-middle attacks etc.) and unwanted leakage of model parameter updates (=*gradients*) which could give an attacker insights into model formation or even allow the reconstruction of input data. In the case of computing embeddings via Graph Representation Learning (e.g. by *neighbourhood aggregation*) the structure of a graph has significant influence on embedding generation. However, to the best of our knowledge, there exist no insights into the correlation of graph topologies and the vulnerability of generated embeddings.

5 Graph Libraries

Since algorithms for graph processing are mostly founded in discrete mathematics, combinatorics and linear algebra, they are usually not very intuitive and therefore hard to program for the uninitiated application programmer. At the heart of all graph computations are therefore suitable graph libraries abstracting away the low-level intricacies of data structures & algorithms, providing instead a high-level API for queries invoking graph traversals, shortest paths, centralities, and even whole recommender pipelines.

Considering the very fundamental & diverse role that graphs play in different application scenarios, graph libraries come in vastly different shapes and sizes - from general-purpose desktop-grade libraries (like Python’s NetworkX) that are easy to use but less performant, to highly specialized implementations running on thousands of CPU/GPU cores; from single-threaded implementations obviating the need to deal with multiple simultaneous read/write operations to streaming graph libraries needing to store potentially infinite graphs.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

Therefore, several issues play a vital role in designing graph libraries [29], some of which shall be discussed here:

1. *Graph type coverage*: Should the library support only certain types of graphs (directed / undirected, weighted / unweighted, typed, with or without node- or edge attributes etc.).
2. *Separation of graph types*: Shall we separate mathematically defined subtypes (directed / un- directed, weighted / unweighted) into different graph classes with only their corresponding well-defined algorithms available on those classes or shall we permit mixed types as well [30]?
3. *Data structures*: Do we implement our graph in the most intuitive way (e.g. objects pointing to each other) lowering the barrier for new programmers to use our API but risking lower performance, or do we optimize our data structures for size / sparsity risking a steeper learning curve (or more abstract API on top of the efficient low-level computational layer)?
4. *Static vs. dynamic*: If a library will be used in real-world scenarios like social networks or product recommendations, queries might get executed against the graph structure while it is continually changing. A suitable library design must therefore provide robust mechanisms to ensure both algorithmic correctness as well as solid performance in simultaneous read/write scenarios [31].
5. *Storage problem*: How to split the edges into smaller units / groups?
6. *Storage distribution*: How can we split large graphs into logical sub-units efficiently so that we can distribute storage across nodes without causing prohibitive communication overhead?
7. *Algorithms*: How can we implement a suite of common algorithms applicable to the widest possible variety of graph problems?
8. *Query optimization*: How do we efficiently query our data-structures w.r.t. to different use cases (runtime performance, constraints, distributed computation)?
9. *Advanced features*: Do we go beyond traditional graph-based database systems by implementing GNN (graph neural network) architectures, distributed computation (like Apache Spark or Hadoop MapReduce) or even edge-computing [32]?
10. *Scalability*: How can we use all the building blocks discussed to scale our graphs to billions of nodes and edges, a requirement in modern web-scale graph problems?
11. *Infinite / implicit graphs*: Do we extend our library to encode potentially infinite graphs (e.g. constructed from an infinite data stream) or graphs of such size that only parts of it will ever be instantiated at any given time (e.g. the state space of a Rubix cube ≥ 3x3).

Since there are too many implementations to be discussed here, figure 12 gives an example of the base architecture of the GBase graph library [29], using graph clustering and block compression as the main building blocks of it’s storage engine.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

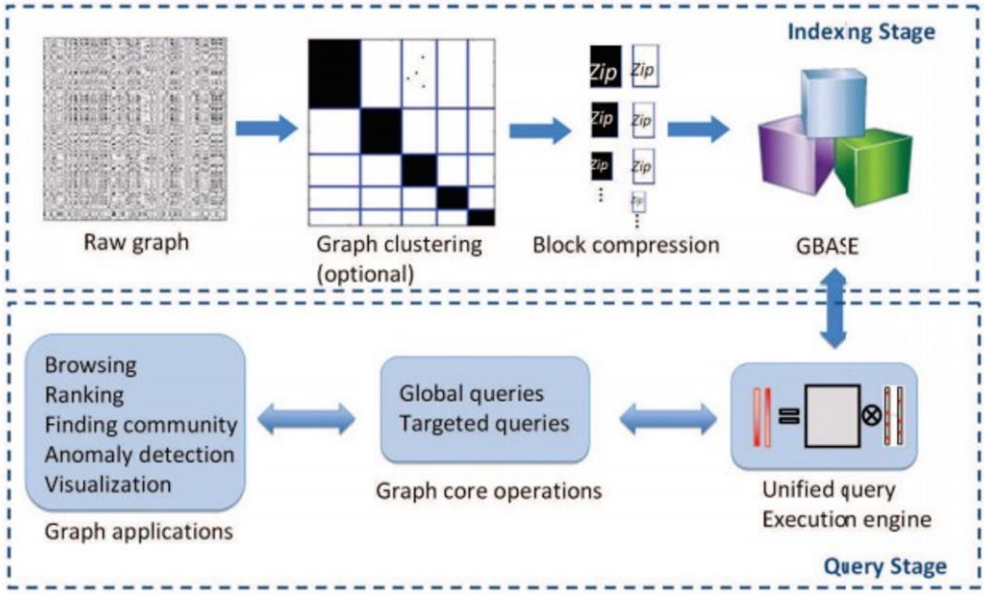


Figure 12 *Overall Framework of GBase core architecture, Source [29]: 1) Indexing Stage: the raw graph is clustered and divided into compressed blocks, 2. Query Stage: global and targeted queries from various graph applications are handled by a unified query execution engine.*

6 Status Quo

We originally expected to gain insights into how different graph topologies influence network representation learning, since most of the available techniques (Deepwalk, node2vec, GraphSAGE etc.) utilize some form of neighbourhood sampling, which strongly depends on the local / global layout of the network. For instance, a random walk in a locally dense graph (e.g. cliques in a social network) could be assumed to stay in the vicinity of a randomly chosen starting node, whereas in a complete graph one could “traverse” the whole graph structure in one single step. Likewise, neighbourhoods sampled from a path graph would have a totally different shape & “meaning” then those sampled from a tessellation graph.

While those hypotheses were certainly pertinent from a theoretical point of view, our practical experience analysing the available data at Medical University Graz guided us to conclude that our power over different representations of the underlying dataset is rather limited. To give an example, we have access to a set of 1.3 million patient cases including diagnoses, which was pre-processed manually and in cooperation with pathologists in order to manually construct and curate a decision tree depicting the generic mental process of medical experts in arriving at a diagnosis. The next step towards automated learning is to model this dataset as a graph & apply representation learning to it, either in an unsupervised fashion, producing human-understandable concept vectors for terms in the database, or in a supervised fashion to learn patient or disease classifications in an end-to-end manner. In order to do this, we generally need to follow a 3-step procedure:

1. Construct an initial graph in accordance with human bias; e.g. if doctors decide that patient age is a relevant factor for diagnosis classification, we would construct edges between patients in same age cohorts.
2. Model the initial feature vectors of nodes in the graph. These can be embeddings themselves, using as inputs carefully selected subsets of available medical data, may it be history, blood pressure, genomics data etc.
3. Use a representation learning neural network architecture to learn feature vector aggregation weights based on the selected inputs & the constructed graph.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

The first step in this process is the most relevant regarding graph types – nevertheless, the resulting structure depends on the bias humans introduce into the network (which is absolutely necessary, because herein lie the assumptions which give algorithms a vantage point to learn from). Therefore, a specific graph type cannot be guaranteed, which would make further research into this field a liability rather than an asset. To exacerbate circumstances, envision the following architectural paradigm, which summarizes our grand vision for the next years (Figure 13):

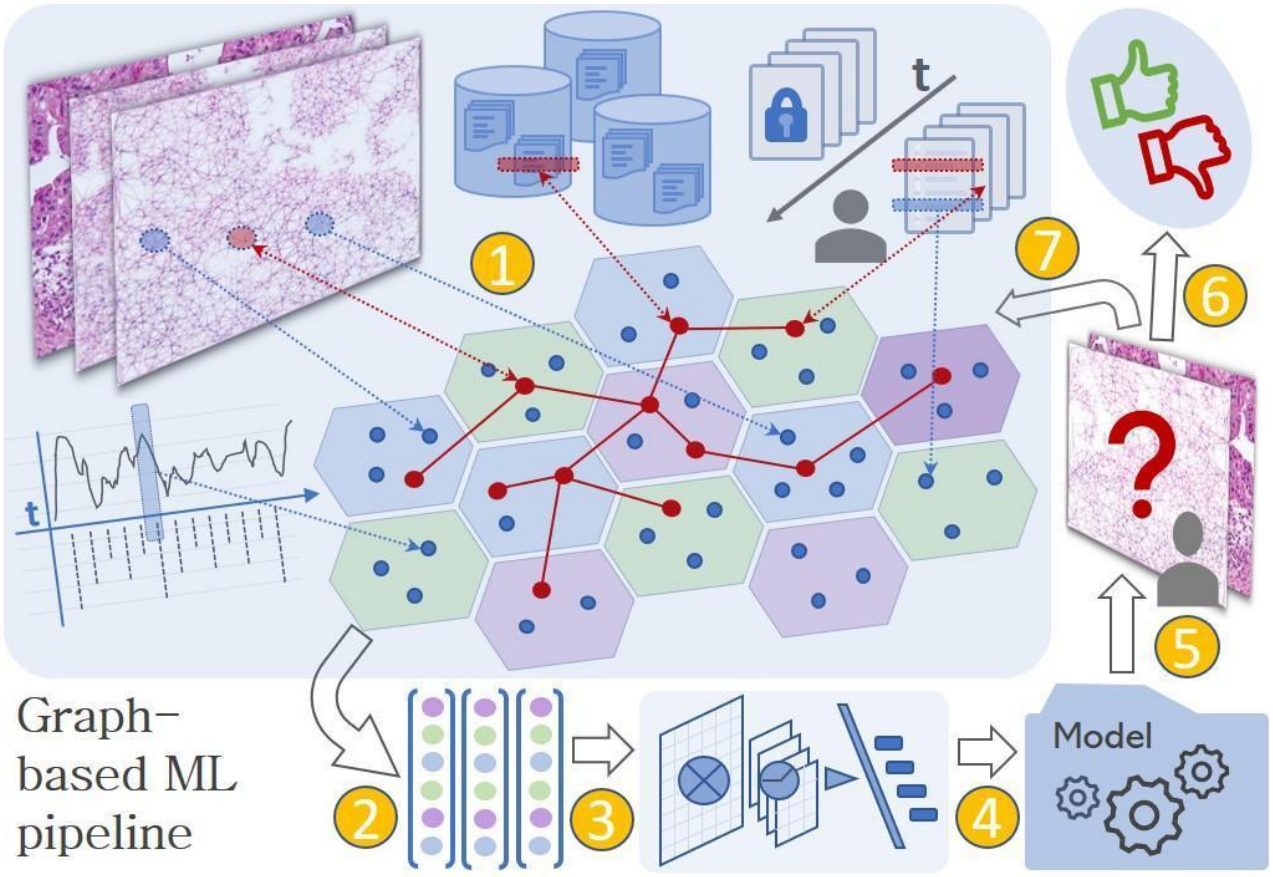


Figure 13: *Our proposal for an integrated, graph-based ML pipeline (Source: Bernd Malle)*

The large blue box labelled “1” illustrates the construction of a decentralized graph (split into tiles) from data originating in several different modalities – a cell graph constructed from histopathological images (this is a current & trending topic in medical AI), entries from a medical knowledge DB, personal case files as well as time series signals (to the very left). The resulting graph will be naturally divided across institutions (= locations of data availability) and will either follow the inherent structure imposed by the underlying data or result from human bias which is crucial in initial graph construction. The resulting multi-modal fusion graph will be naturally distributed but resist classification into pure, particular text-book types. Therefore we foresee that we won’t (and not be able to) follow traditional graph partitioning methods but rather let reality guide us in the design of this distributed computing paradigm. Our current insights suggest message-passing methods inherent in graph convolution can be adapted to decentralized approaches and even extended to include privacy-aware ML, which we will outline in our next report *D4.4 “Experimental results for shape and composition of connection surfaces”* (also see the updated version of our graph parallelism report (D4.1)).

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

7 Conclusion

In this report, we provided a very subjective overview of graph classes we deem relevant for our work in the project FeatureCloud as well as criteria for designing a useful graph library, which are vital in order to design the right platform for the right task. Moreover, we conducted research and data analysis tasks on a real-world medical dataset of 1.3 million patients, and laid the theoretical framework for understanding how current techniques in network representation learning could be adapted to the distributed case. The combination of insights gained leads us to the conclusion that striving to model graphs according to specific, text-book types would be impractical and actually hinder our work.

Thus, the next steps in our endeavours will be to

1. apply our understanding of graphs & their structural properties to a series of network representation learning experiments that we outlined in our report on *graph parallelism (D4.1),*
2. to experiment with several, differently initialized graphs according to human bias in the field of medical patient records,
3. run experiments with *Graph Neural Networks* to see if human bias (and if yes, which assumptions) in constructing an initial graph will lead to higher accuracy in results.
4. to compare these two methods of initial graph construction without human bias (e.g. by merely employing the underlying text corpus or its embeddings).
5. to model current message passing algorithms to work on a decentralized graph; most importantly to reduce communication overhead by computing subgraph-specific representative vectors. This would help a lot in building an adaptive swarm of graph-based *local spheres* whose exact shape at runtime cannot be foreseen AND should remain private to the respective local agent.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

8 References

[1] John M. Mellor-Crummey and John Garvin. Optimizing sparse matrix-vector product computations using unroll and jam. International Journal of High Performance Computing Applications, 18(2 SPEC. ISS.): 225–236, 2004. ISSN 10943420. doi: 10.1177/1094342004038951.

[2] CardiffUniversity(graphtheory).http://users.cs.cf.ac.uk/Dave.Marshall/CM0167/Lectures/CM0167_Chap02_1_Trees.pdf [Online; accessed 12-22-2019].

[3] D. G. Corneil and C. C. Gotlieb. An Efficient Algorithm for Graph Isomorphism. Journal of the ACM (JACM), 17(1):51–64, 1970. ISSN 1557735X. doi: 10.1145/321556.321562.

[4] László Babai, Paul Erdos, and Stanley M. Selkow. Random Graph Isomorphism. SIAM Journal on Computing, 9(3):628–635, 1980. ISSN 0097-5397. doi: 10.1137/0209047.

[5] Richard C. Bruce. An Explanation for Differences in Body Size between Two Desmognathine Salamanders. Copeia, 1990(1):1, 1990. ISSN 00458511. doi: 10.2307/1445815.

[6] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub)graph isomorphism algorithm for matching large graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(10): 1367–1372, 2004. ISSN 01628828. doi: 10.1109/TPAMI.2004.75.

[7] László Babai. Graph isomorphism in quasipolynomial time: [Extended abstract]. Proceedings of the Annual ACM Symposium on Theory of Computing, 19-21-June-2016:684–697, 2016. ISSN 07378017. doi: 10.1145/2897518.2897542.

[8] M. Voigt, List colourings of planar graphs, Discrete Mathematics 306 (10- 11) (2006) 1076–1079. doi:10.1016/j.disc.2006.03.027.

[9] R. C. Bose. Strongly regular graphs, partial geometries and partially balanced designs. Pacific J. Math., 13 (2):389–419, 1963. URL https://projecteuclid.org:443/euclid.pjm/1103035734.

[10] Markus Meringer. Fast generation of regular graphs and construction of cages. Journal of Graph Theory, 30(2):137–146, 1999. ISSN 03649024. doi: 10.1002/(SICI)1097-0118(199902)30:2<137:: AID-JGT7>3.0.CO;2-G.

[11] Peter J Cameron. Strongly regular graphs An example. pages 1–23, 2001.

[12] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. CoRR, abs/1403.6652, 2014. URL http://arxiv.org/abs/1403.6652.

[13] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. CoRR, abs/1605.05273, 2016. URL http://arxiv.org/abs/1605.05273.

[14] Béla Bollobás, Christian Borgs, Jennifer Chayes, and Oliver Riordan. Directed scale-free graphs. Proceed- ings of the Annual ACM-SIAM Symposium on Discrete Algorithms, pages 132–139, 2003.

[15] Béla Bollobás and Oliver M. Riordan. Mathematical results on scale-free random graphs. Handbook of Graphs and Networks, pages 1–34, 2004. doi: 10.1002/3527602755.ch1.

[16] Béla Bollobás and Oliver Riordan. Robustness and Vulnerability of scale-free random graphs. Internet Mathematics, 1(1):1–35, 2004. ISSN 19449488. doi: 10.1080/15427951.2004.10129080.

[17] Albert László Barabási and Réka Albert. Emergence of scaling in random networks. Science, 286(5439): 509–512, 1999. ISSN 00368075. doi: 10.1126/science.286.5439.509.

[18] Reuven Cohen and Shlomo Havlin. Scale-Free Networks Are Ultrasmall. Physical Review Letters, 90(5): 4, 2003. ISSN 10797114. doi: 10.1103/PhysRevLett.90.058701.

[19] Roger Pearce, Maya Gokhale, and Nancy M. Amato. Scaling techniques for massive scale-free graphs in distributed (external) memory. Proceedings - IEEE 27th International Parallel and Distributed Processing Symposium, IPDPS 2013, pages 825–836, 2013. doi: 10.1109/IPDPS.2013.72.

[20] Lun Li, David Alderson, John C. Doyle, and Walter Willinger. Towards a theory of scale-free graphs: Definition, properties, and implications. Internet Mathematics, 2(4):431–523, 2005. ISSN 15427951. doi: 10.1080/15427951.2005.10129111.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.

[21] L. Lovász. Normal hypergraphs and the perfect graph conjecture. Discrete Mathematics, 2(3):253–267, 1972. ISSN 0012365X. doi: 10.1016/0012-365X(72)90006-4.
[22] Joshua Cooper and Aaron Dutle. Spectra of uniform hypergraphs. Linear Algebra and Its Applications, 436(9):3268–3292, 2012. ISSN 00243795. doi: 10.1016/j.laa.2011.11.018. URL http://dx.doi.org/ 10.1016/j.laa.2011.11.018.
[23] Lei Zhou, Sandy El Helou, Laurent Moccozet, Laurent Opprecht, Omar Benkacem, Christophe Salzmann, and Denis Gillet. A federated recommender system for online learning environments. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7558 LNCS:89–98, 2012. ISSN 03029743. doi: 10.1007/978-3-642-33642-3_10.
[24] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and ap- plications. Discrete Applied Mathematics, 42(2-3):177–201, 1993. ISSN 0166218X. doi: 10.1016/ 0166-218X(93)90045-P.
[25] Paul Erdos and Alfred Renyi. , P„ by selecting N edges from the ; i) possible edges Px (1 5. Publicationes Mathematicae, 6:290–297, 1959.
[26] Mohammad Mahdian. Stochastic Kronecker Graphs. (1):1–9.
[27] Jure Leskovec Kronecker graphs. http://www.cs.cmu.edu/~jure/pub/ kronecker-cornell-Sept08.pdf, 2008. [Online; accessed 12-22-2019].
[28] Walter Schnyder. Planar Graphs on the Grid.
[29] U. Kang, Hanghang Tong, Jimeng Sun, Ching Yung Lin, and Christos Faloutsos. GBASE: A Scalable and general graph management system. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1091–1099, 2011. doi: 10.1145/2020408.2020580.
[30] Bernd Malle. Graphinius - a web based graph exploration and analysis platform. http://diglib.tugraz. at/download.php?id=5990d1fbbd8d4&location=browse, 2016. [Online; accessed 12-22-2019].
[31] Jure Leskovec and Rok Sosicˇ. Snap: A general-purpose network analysis and graph-mining library. ACM Trans. Intell. Syst. Technol., 8(1):1:1–1:20, July 2016. ISSN 2157-6904. doi: 10.1145/2898361. URL http://doi.acm.org/10.1145/2898361.
[32] Bernd Malle, Nicola Giuliani, Peter Kieseberg, and Andreas Holzinger. The More the Merrier - Federated Learning from Local Sphere Recommendations. In LNCS 10410 of Machine Learning and Knowledge Extraction, pages 367–373, Reggio, Italy, 2017. Springer International Publishing. doi: 10.1007/978-3-319-66808-6_24. URL https://hal.inria.fr/hal-01677145.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 826078.