Anomaly detection on graphs

Benoît Corsini

joint work with P.-A. Noël, D. Vázquez, & P. Taslakian

- Problem
- Method
- Algorithms
- Results

- Problem
- Method
- Algorithms
- Results

Anomaly detection in graphs

We are trying to solve **unsupervised anomaly detection** on graphs:

- **Our input** is a graph where the nodes have a given set of features.
- **Our output** can either be a *binary labelling* or a *ranking* of the nodes.

The output should reflect some sense of **anomalousness** of the nodes.

Why is this problem interesting?

- Relates to **bot identification** in social networks.
- Can help improve **spam detection**.
- Relates to various types of **fraud**: *insurance*, *healthcare*, *financial*, etc.

➔ A good anomaly detection method could also help **other graph-based tasks**, such as *classification* or *clustering*.

Defining anomalies

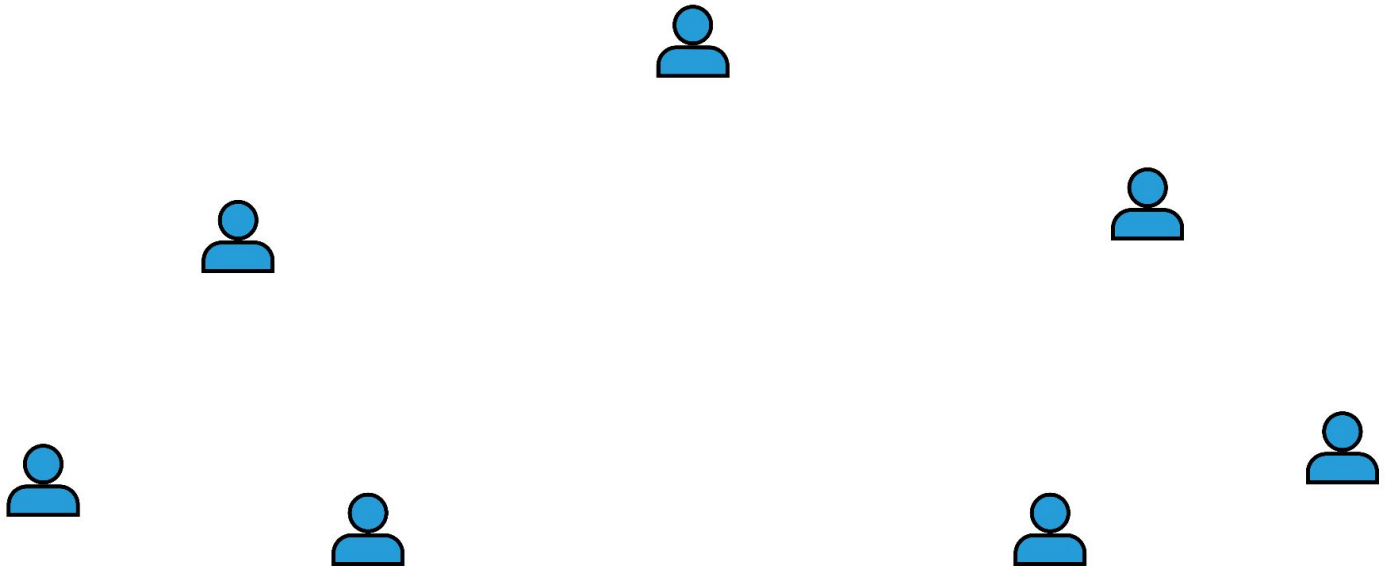
Defining anomalies on graphs is a **difficult task** since nodes have **two types of information**:

- *Personal information*, given by their **features**.
- *Community information*, given by their **neighbours**.

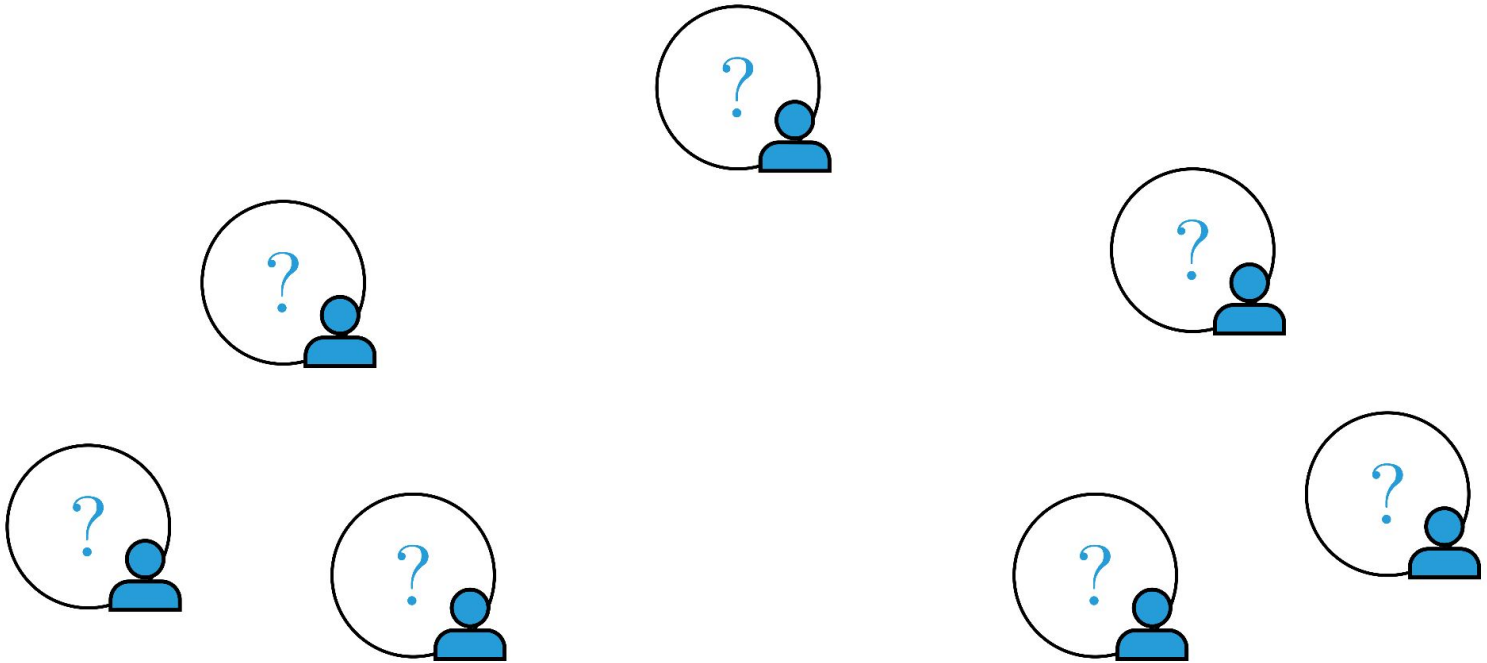
In that context, anomalies can have several forms:

- **Feature-based anomalies:** an anomaly can be defined uniquely *based on its features*.
- **Graph-based anomalies:** an anomaly can be defined uniquely *based on its neighbours* and the general graph structure.
- **Combined anomalies:** an anomaly can be defined not in the two previous ways, but by considering *both its features and the general graph structure*.

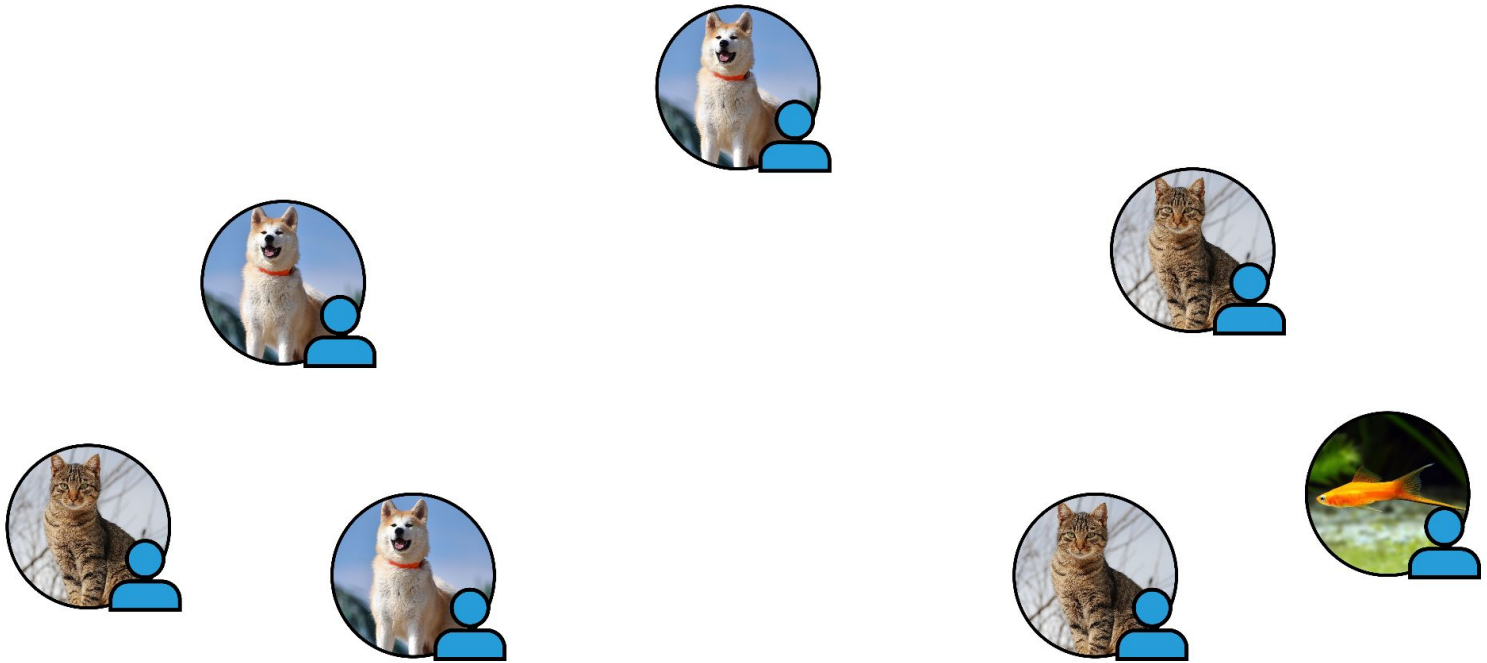
Defining anomalies (an example)



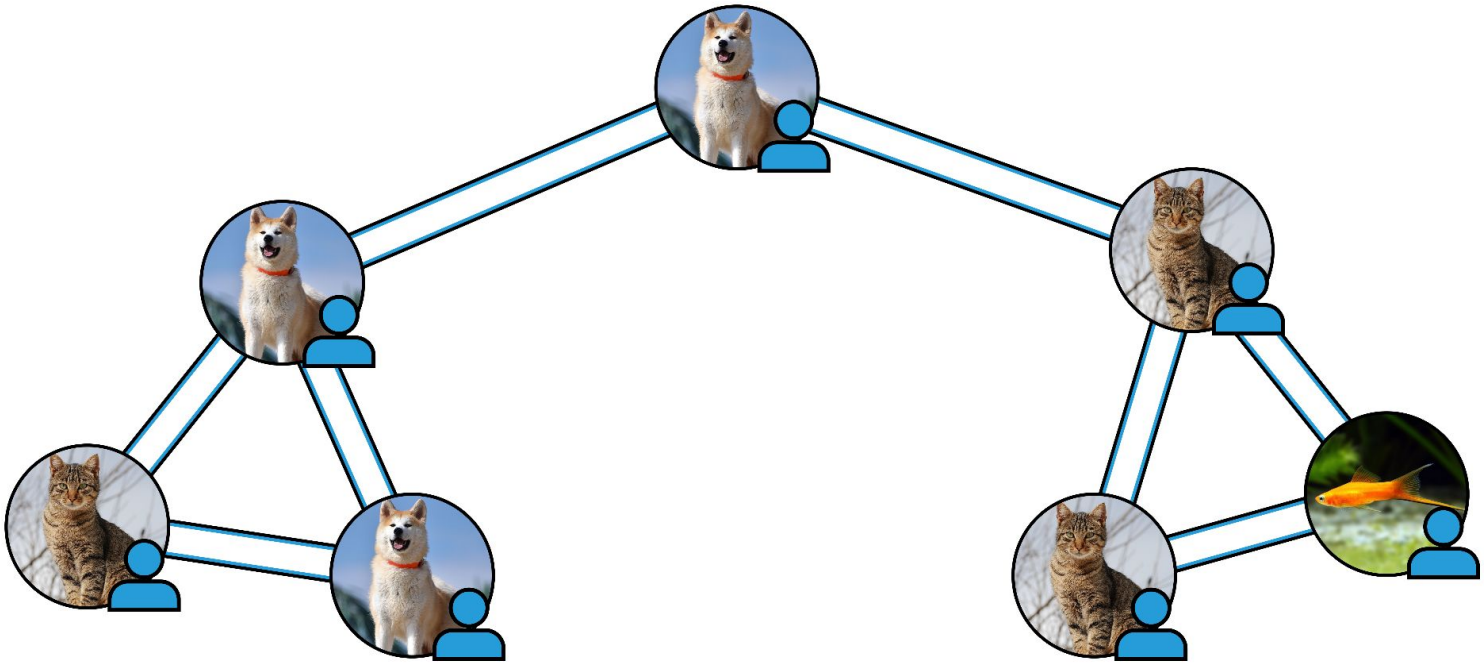
Defining anomalies (an example)



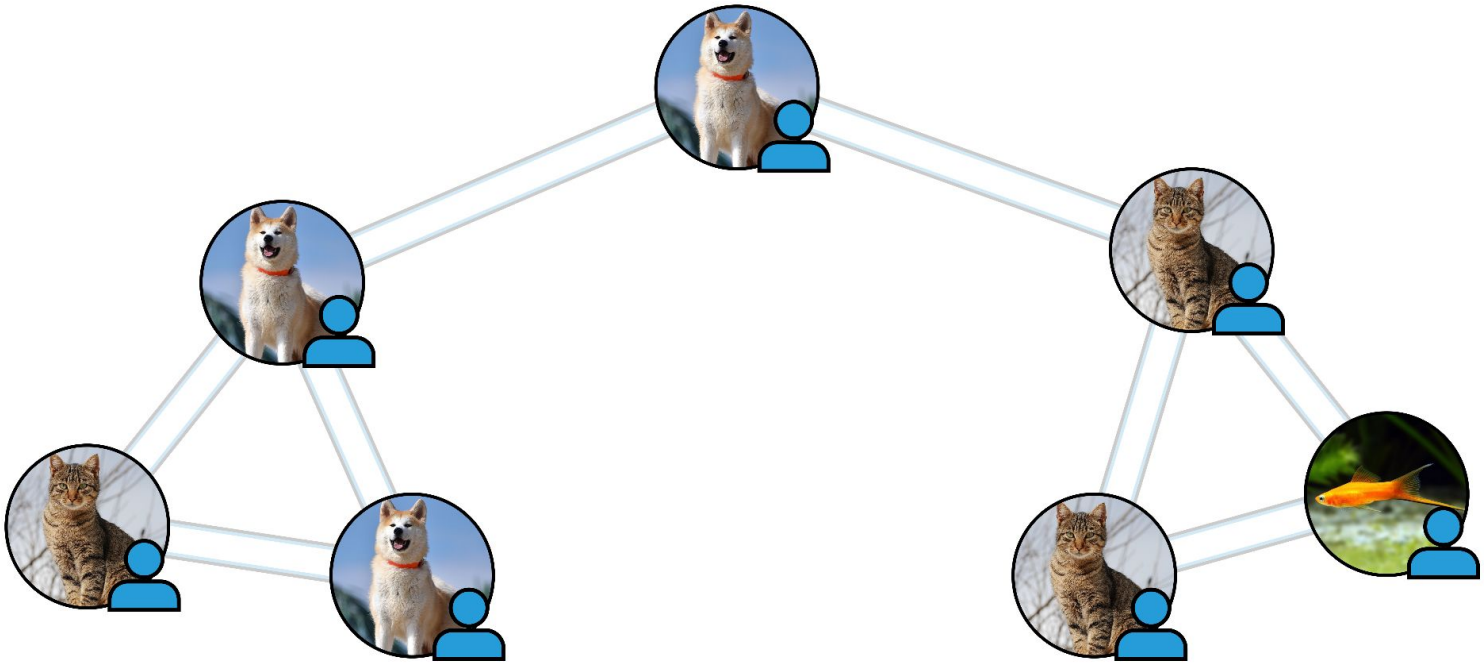
Defining anomalies (an example)



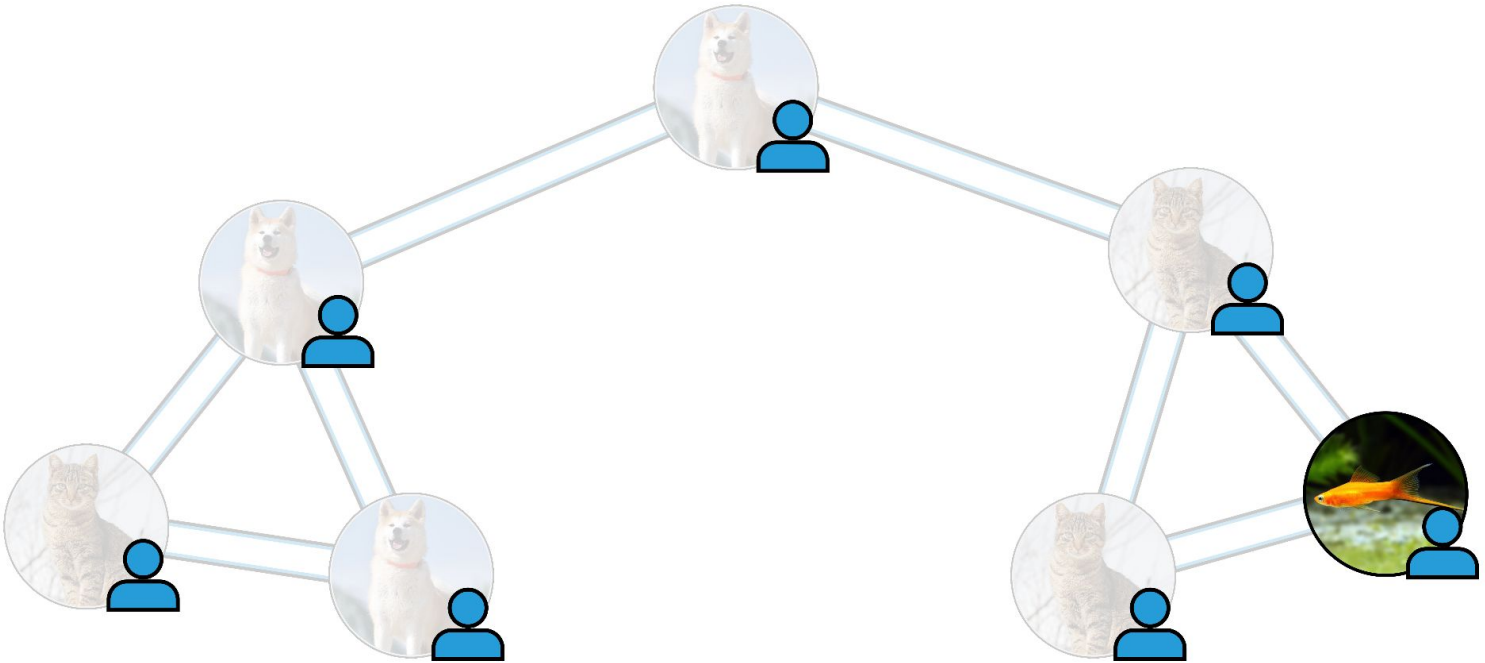
Defining anomalies (an example)



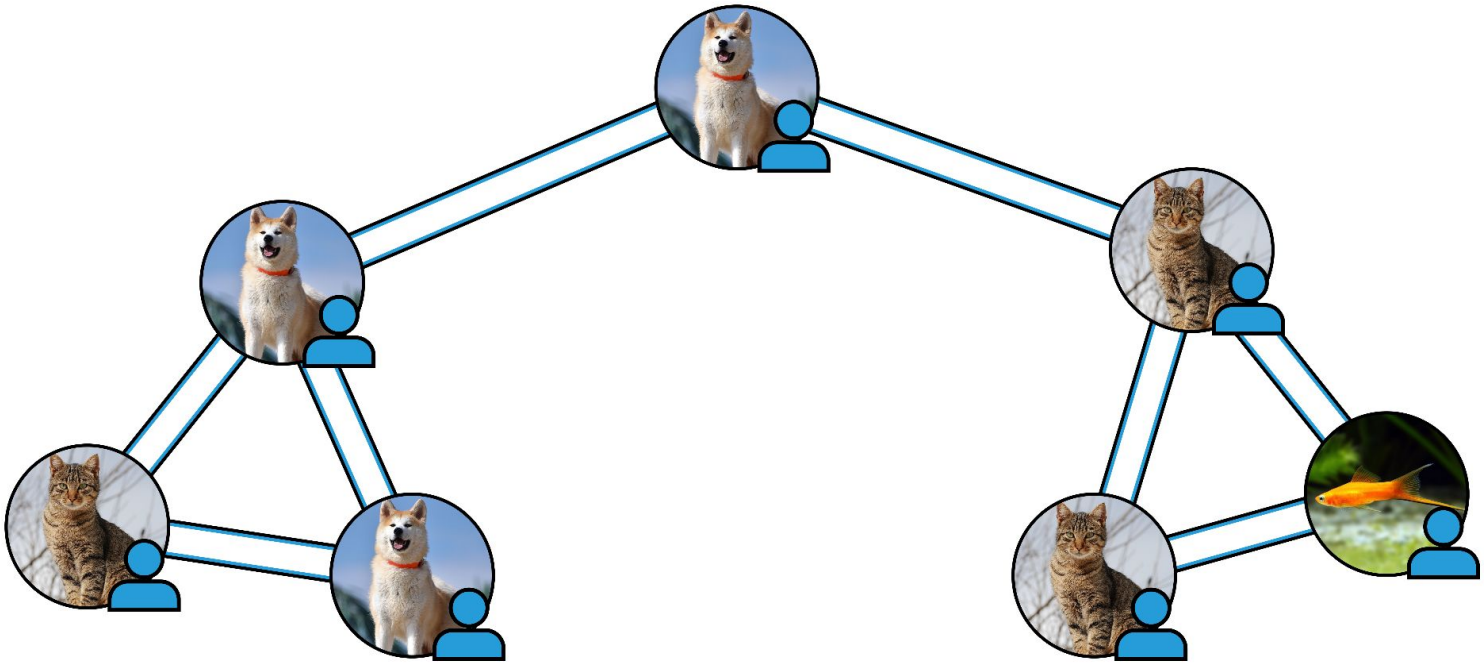
Defining anomalies (an example)



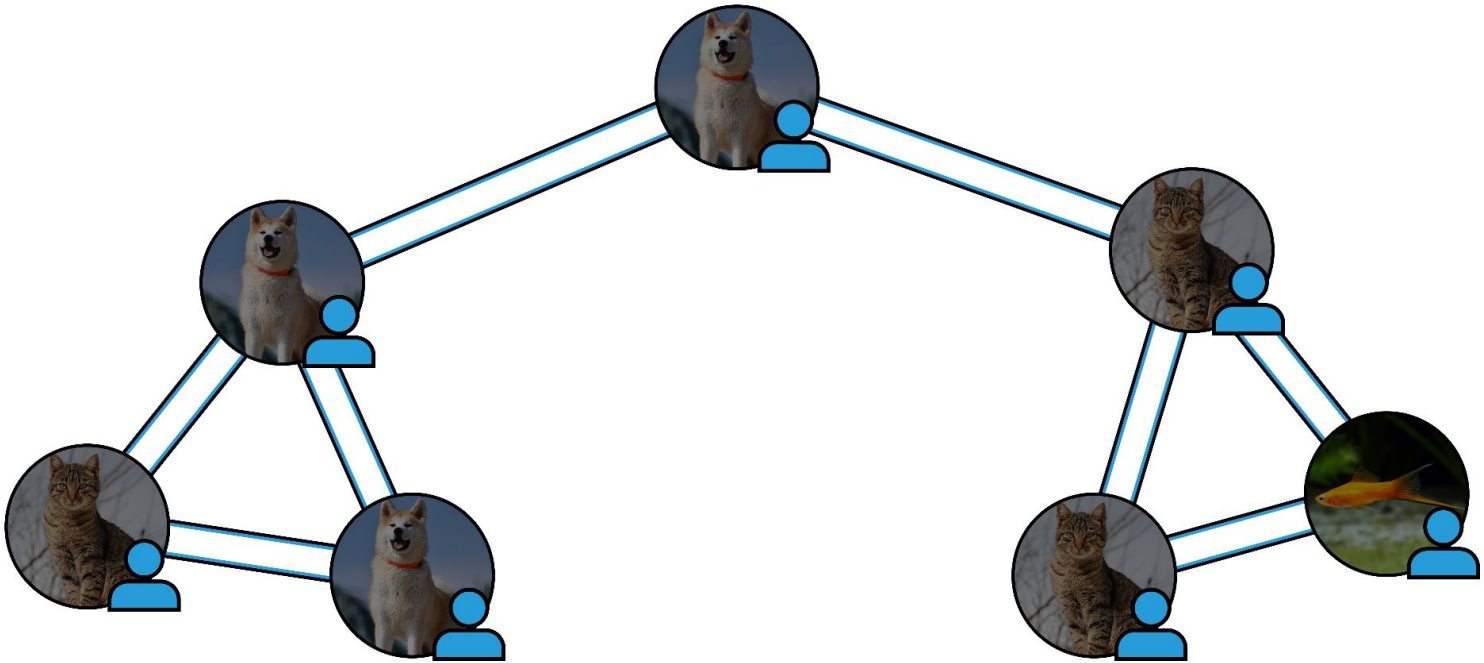
Defining anomalies (an example)



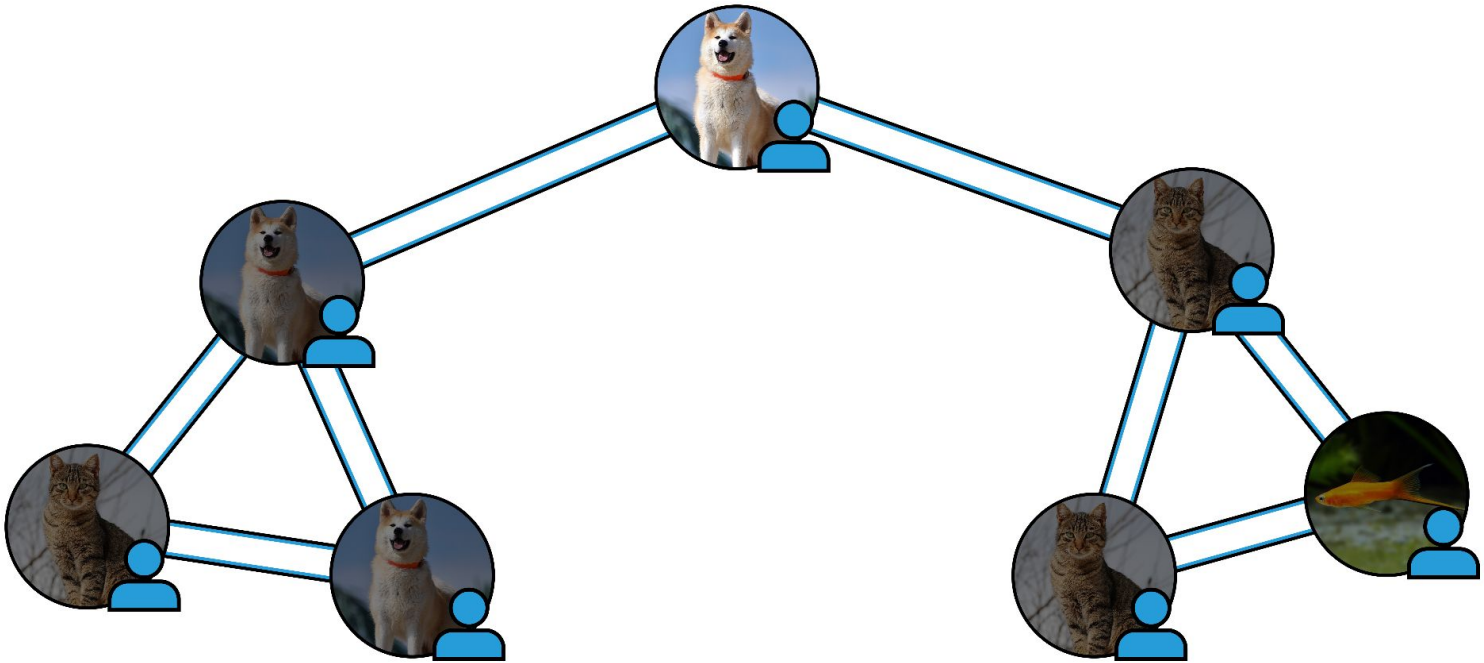
Defining anomalies (an example)



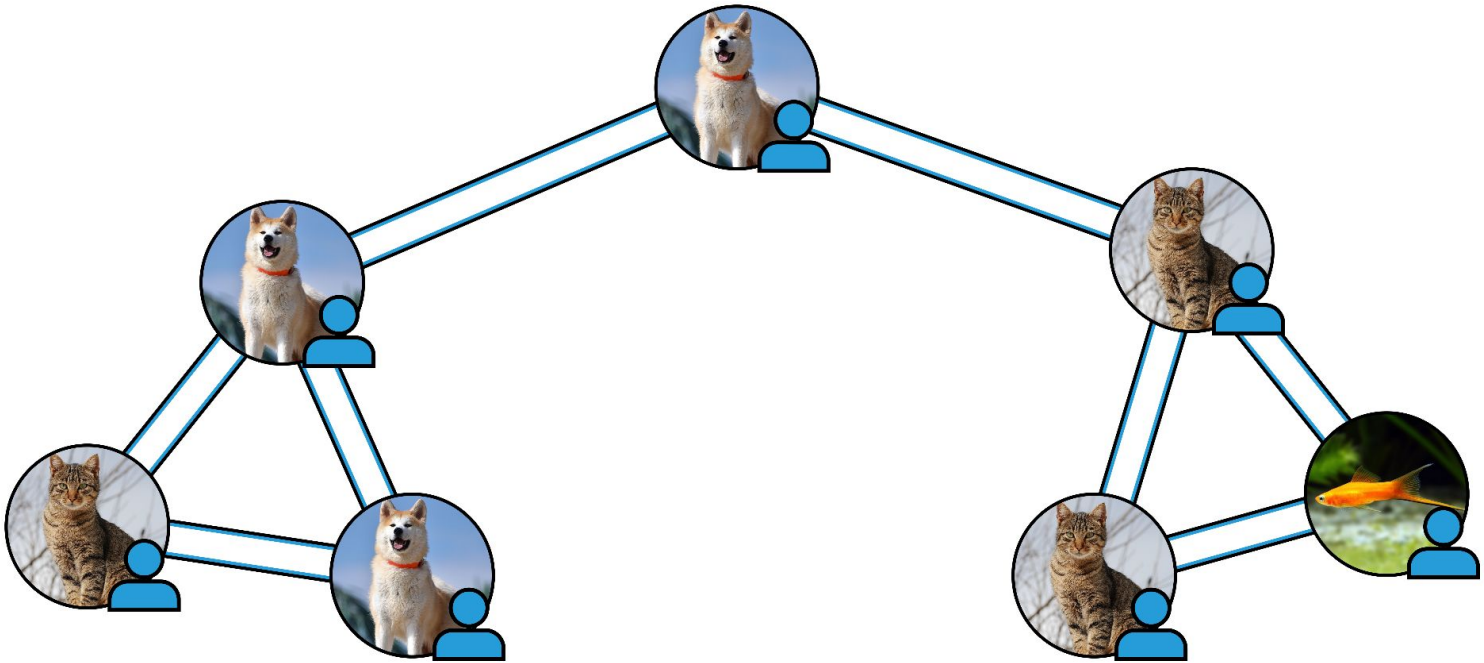
Defining anomalies (an example)



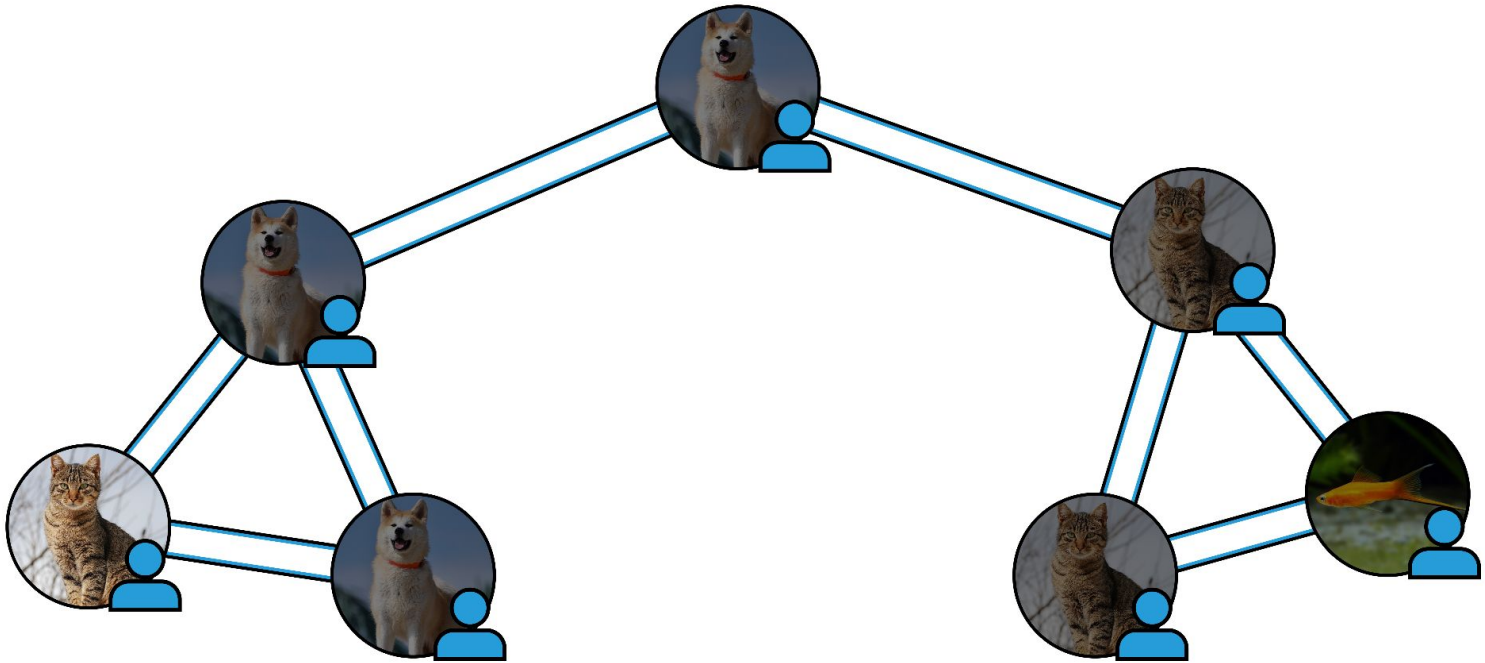
Defining anomalies (an example)



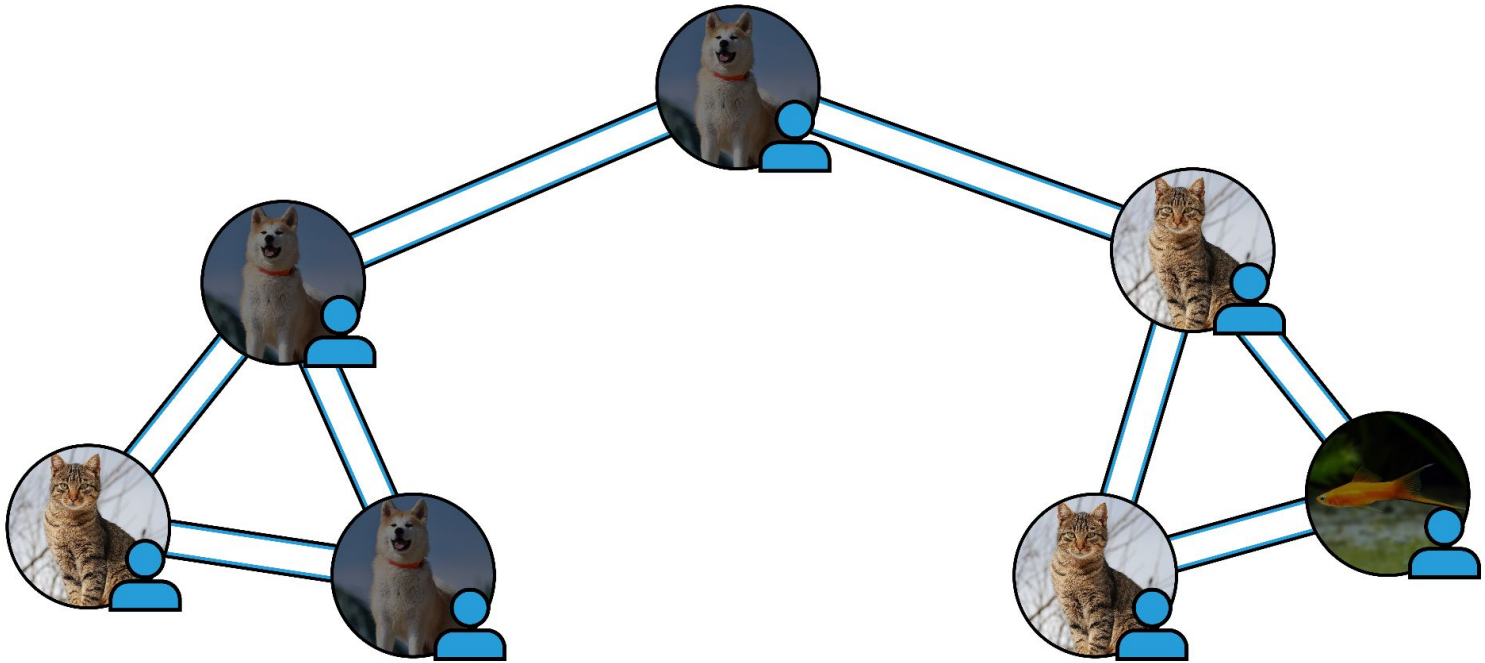
Defining anomalies (an example)



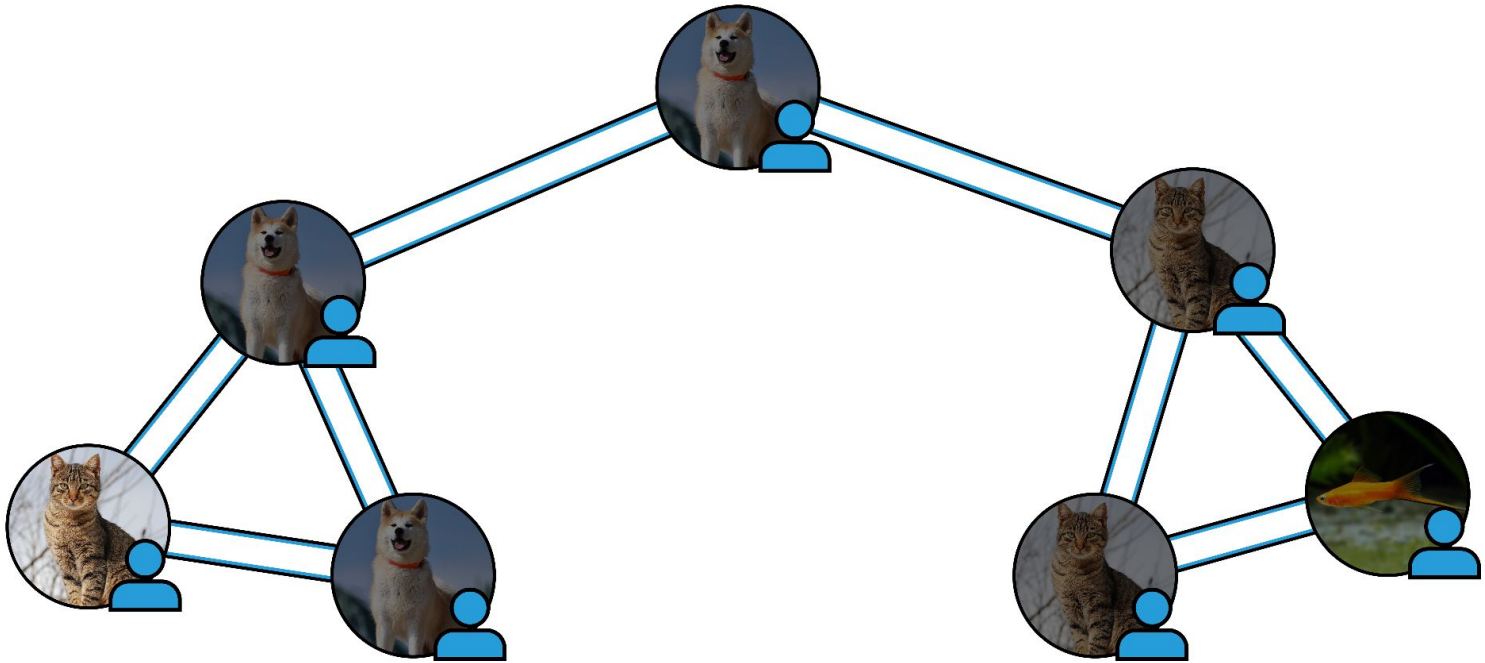
Defining anomalies (an example)



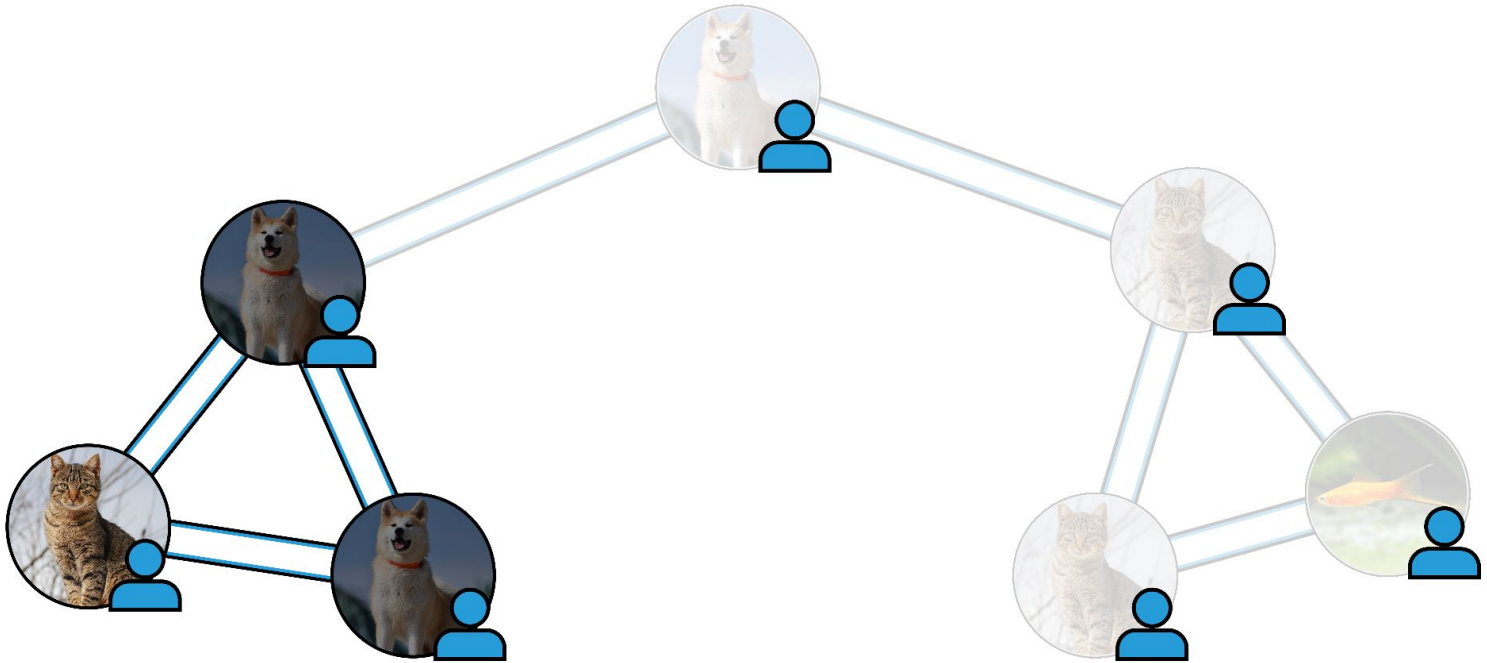
Defining anomalies (an example)



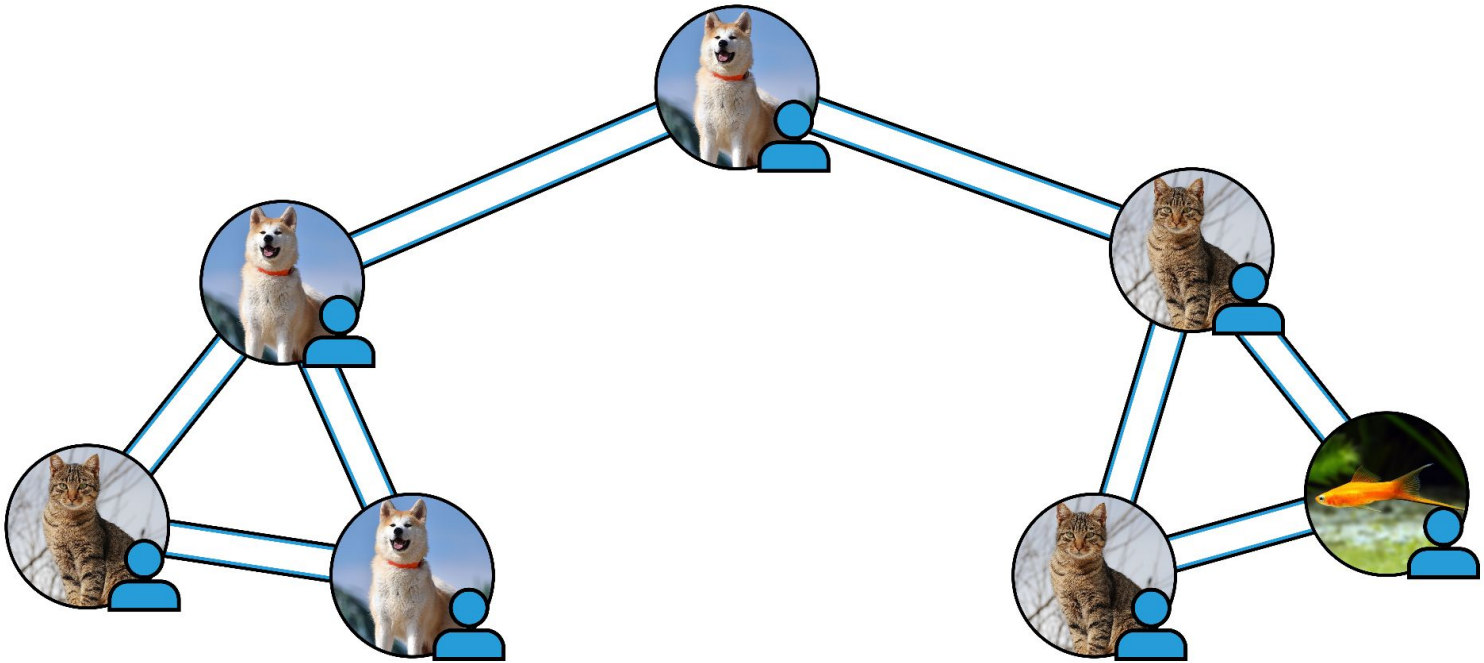
Defining anomalies (an example)



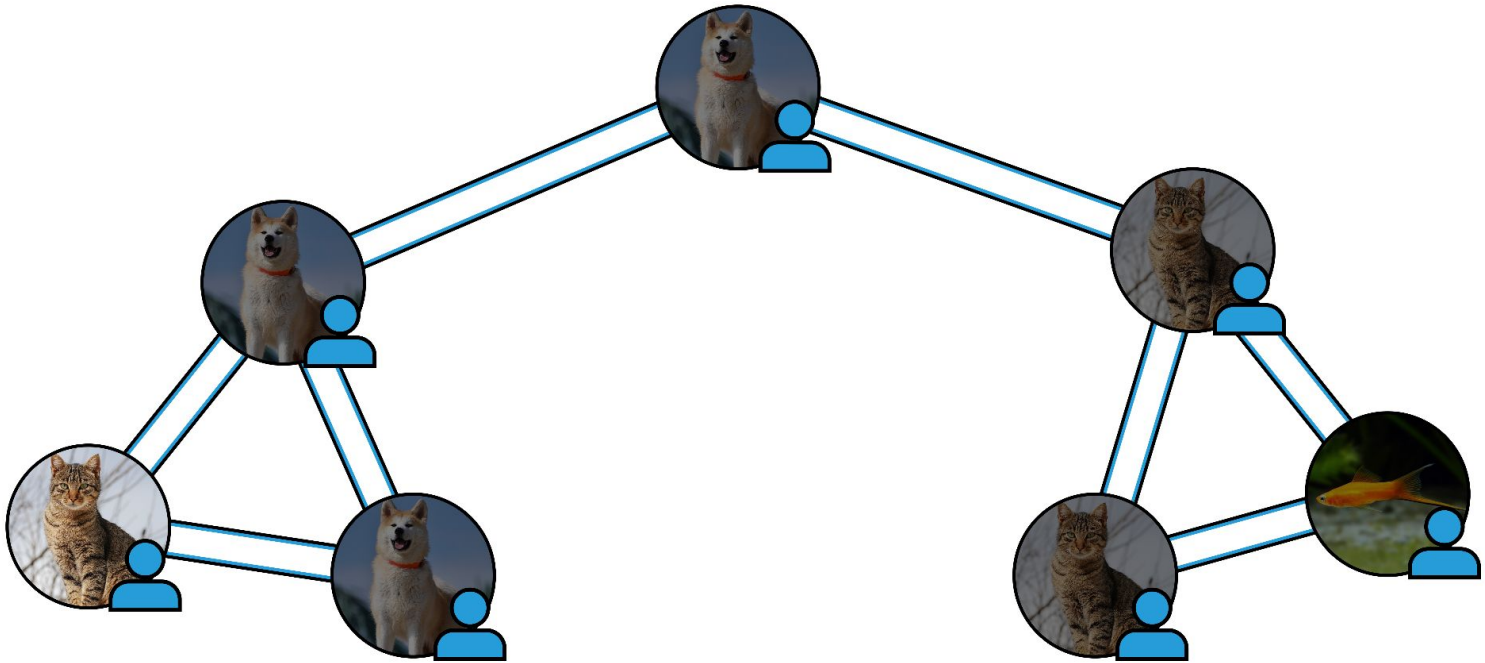
Defining anomalies (an example)



Defining anomalies (an example)



Defining anomalies (an example)



Previous work

The literature in anomaly detection on graphs is very **diverse** and **sparse**.

We did not find **direct baselines** and had to base our work on the following articles.

- **Multi-scale Anomaly Detection on Attribute Networks**, *L. Gutiérrez-Goméz, A. Bovet, J.-C. Delvenne*, 2019
- **Bayesian Robust Attributed Graph Clustering: Joint Learning of Partial Anomalies and Group Structure**, *A. Bojchevski, S. Günnemann*, 2018
- **Maximum Entropy Generators for Energy-Based Models**, *R. Kumar, S. Ozair, A. Goyal, A. Courville, Y. Bengio*, 2019

- Problem
- Method
- Algorithms
- Results

General approach

Since we are using an **unsupervised method**, we do not have access to labels or examples of anomalies.

This is a realistic approach since:

- Anomalies are **rare**.
- Anomalies might be **very diverse**.

➔ The only assumption we can make is that **most nodes are normal**.

With these constraints, the **general approach** to anomaly detection is to:

- Define some **energy** of the nodes, related to their *likelihood of normality*.
- Use this energy to identify anomalies:
  - by **ranking** the nodes;
  - by considering the nodes with **highest energy**;
  - by identifying a **threshold** in the energy distribution.

Our method

The idea:

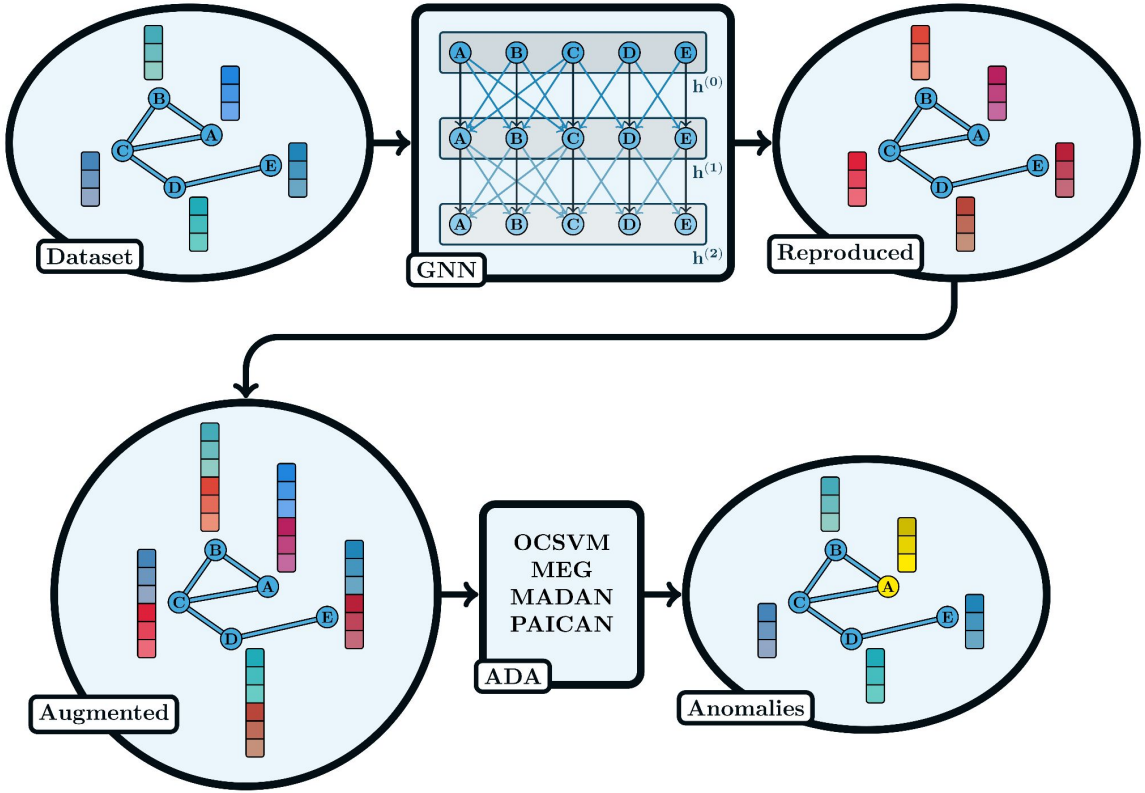
- We will **reproduce** the original features of the nodes with their expected values.
- We first **hide/mask** some of the features and then try to recreate them using the rest of the information.
- Use *Graph Neural Networks* (**GNNs**) for their high representative power.
- **Compare** the reproduced features with the original ones to identify anomalies.

➔ Creation of two algorithms: **HideGNN** and **MaskGNN**.

The general method:

- Take the graph as input, with node features.
- Use one of our two algorithms (**HideGNN** or **MaskGNN**) to reproduce the node features.
- Use these reproduced features to **augment the original dataset**.
- Apply an **anomaly detection algorithm** (ADA) on the augmented dataset.

Our method


Dataset
GNN
h(0)
h(1)
h(2)
Reproduced
Augmented
OCSVM
MEG
MADAN
PAICAN
ADA
Anomalies

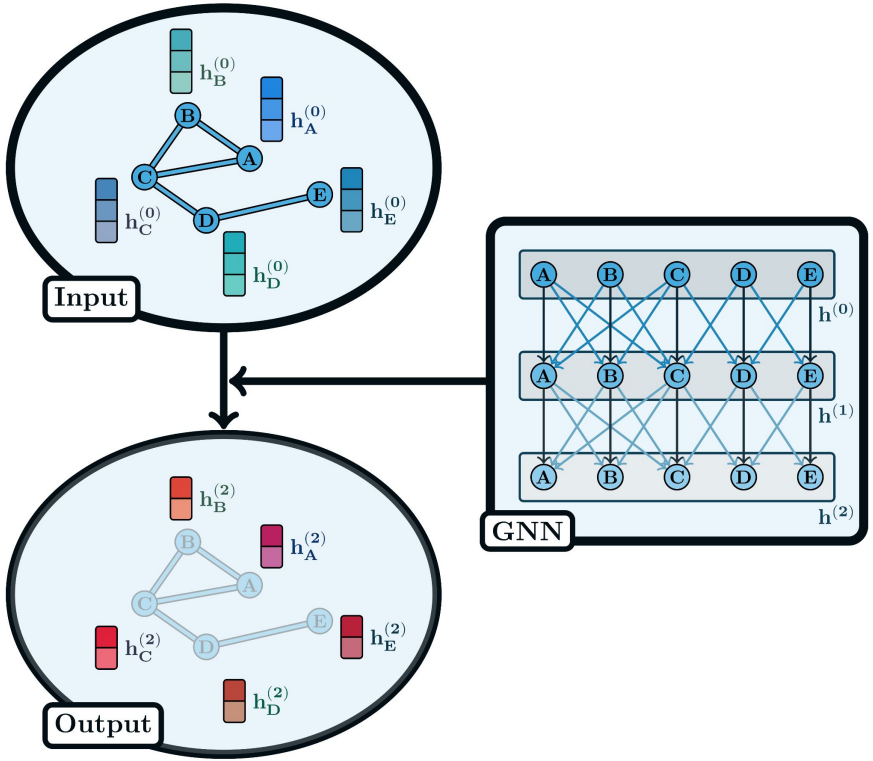
- Problem
- Method
- Algorithms
- Results

GNNs

GNNs are **commonly used** in machine learning methods on graphs.

They are based on a **message-passing** algorithm and have the advantage to:

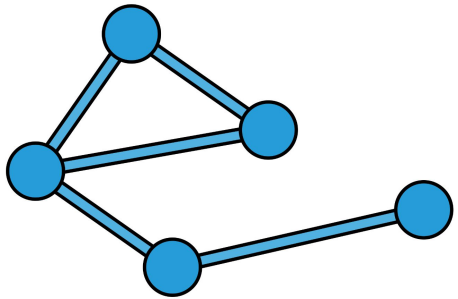
- use **both** the node features and the graph structure;
- and **not depend** on the ordering of the nodes.



HideGNN

The method:

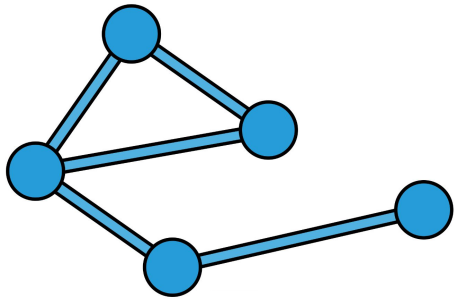
- Start by **hiding** a specific feature.
- Use a GNN on the **other features** to guess the hidden one.
- **Repeat** over all features.
- Eventually obtain **a set of GNNs** able to reproduce the original features.



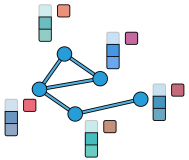
MaskGNN

The method:

- Start by creating a **random mask** over all features.
- **Replace** the masked features by some value (usually 0).
- Use a GNN to reproduce all features but **only optimize on the masked features**.
- **Resample** the random mask.
- Eventually obtain a **non-trivial GNN** which reproduces the original features.

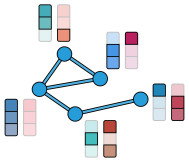


Pros and Cons



HideGNN

- **Pros:**
  - Complete reproduction of the features.
  - Less parameters.
- **Cons:**
  - Heavier to train.
  - Do not use any information from the hidden feature.



MaskGNN

- **Pros:**
  - Faster to train and using a single GNN.
  - Elegant all-in-one tool.
- **Cons:**
  - More parameters.
  - If not properly parametrized, likely to miss some information.
  - Less stable.

Remarks

On our algorithms:

- They are theoretically able to **overfit** and recreate the original features of the dataset.
- They learn from both the graph structure and the features, and their output contains a **lot of information** from the graph.
- They use a novel method combining GNN with methods from **NLP**, which opens the door to new parallel between the two fields.

On our method:

- It allows for **non graph-based algorithms** to be efficiently applied on graphs.
- It could easily be used on **other tasks**.

- Problem
- Method
- Algorithms
- Results

Datasets

We tested our models on **three datasets**:

- **Disney:** 124 nodes and 6 anomalies.
- **Books:** 1418 nodes and 28 anomalies.
- **Enron:** 13533 nodes and 5 anomalies.

The inherent **extreme statistics** of these datasets makes this task complicated.

➔ Only Books allows an *acceptable* split into **train/valid/test**.

	Disney	Books	Enron
Nodes	124	1418	13533
Edges	335	3695	176987
Features	32	21	18
Outliers	6	28	5

Experimental setup

- Since only Books can be split, we use this dataset to **optimize** our algorithms.
- Once we finish running the algorithms, we use parameters with the best results on **valid of Books**.
- We then compute and **report** the results on the test set of Books and on Disney and Enron.

➔ We compare the ADAs **with or without** using our algorithms.

Some problems we encountered:

- None of these datasets were **already split** before our experiments.
- The results previously reported seem to be using the **test set** for hyperparameter tuning.
- We re-computed these results using our framework and found great **discrepancies**. We also did not always manage to recreate their experiments.
- These previous methods also seemed to be using a **modified version** of the datasets, with less features.

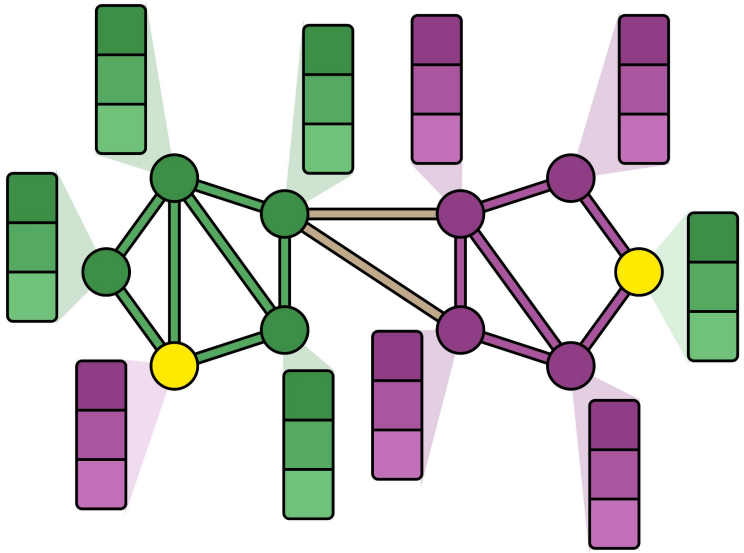
Our results

	Books		Disney*		Enron*	
	Original	Reduced	Original	Reduced	Original	Reduced
OCSVM†	35.90‡	37.33‡	36.44‡	**85.88**‡	55.90‡	43.22‡
Hide + OCSVM	49.58 (±1.47)	**52.21** (±2.41)	46.97 (±2.26)	43.16 (±1.03)	32.76 (±3.79)	**51.84** (±3.35)
Mask + OCSVM	**55.06** (±1.64)	48.49 (±2.07)	**47.63** (±1.73)	43.07 (±6.30)	**56.28** (±9.73)	50.46 (±6.55)
MEG† [24]	**63.12**‡	56.60‡	50.28‡	39.55‡	26.47‡	**63.08**‡
Hide + MEG	60.42 (±7.03)	**65.36** (±3.26)	46.51 (±6.51)	**69.44** (±3.47)	**44.22** (±14.05)	40.12 (±6.54)
Mask + MEG	57.06 (±6.55)	62.17 (±4.67)	**51.25** (±3.46)	62.44 (±13.38)	42.36 (±16.82)	40.63 (±12.21)
MADAN [13]	48.80‡	45.54‡	**67.51**‡	19.49‡	**61.81**‡	**73.64**‡
Hide + MADAN	**57.21** (±9.26)	52.08 (±7.64)	59.09 (±15.03)	**49.82** (±15.02)	54.24 (±14.41)	47.47 (±13.33)
Mask + MADAN	58.43 (±8.87)	**52.34** (±7.89)	55.98 (±14.07)	48.87 (±12.54)	60.55 (±14.58)	50.20 (±14.13)
PAICAN [2]	67.76‡	55.38‡	**75.14**‡	**73.45**‡	26.94‡	57.77‡
Hide + PAICAN	68.23 (±1.45)	**72.82** (±3.38)	62.73 (±8.14)	71.92 (±1.15)	**54.59** (±3.31)	58.65 (±10.07)
Mask + PAICAN	**68.90** (±1.15)	61.25 (±3.43)	67.31 (±6.87)	64.76 (±10.82)	45.92 (±8.08)	**65.40** (±5.46)

A synthetic dataset (Thanksgiving)

To try and understand our algorithms and results better, we created a **synthetic dataset**:

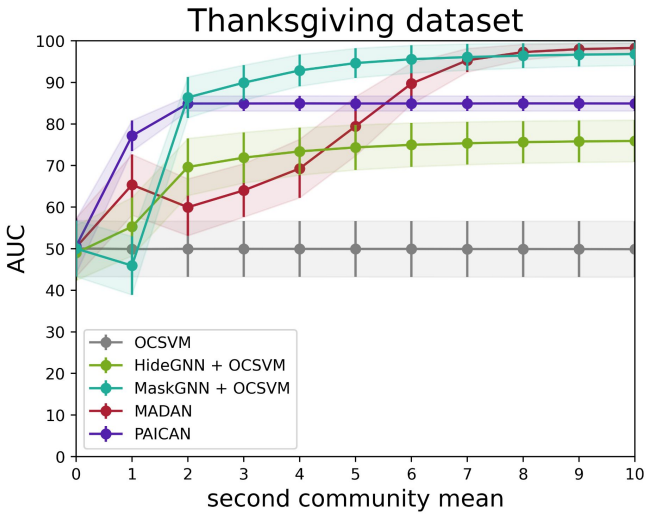
- Composed of **two communities**;
- Each community having a **given set of features**.
- We then modify the features of some members of one community to have the **features of the other community**.



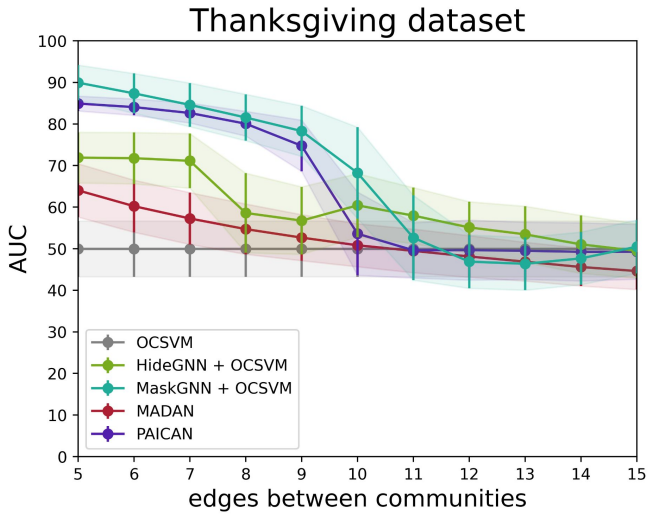
➔ This dataset only has **combined anomalies** and requires to use both the features and the graph structure.

Results on the synthetic dataset

Modifying the difference of **mean** between the two community features:



Modifying the **number of connections** between the two groups:



Thank you!