Beyond Frequencies: **Graph Pattern Mining in Multi-weighted Graphs**

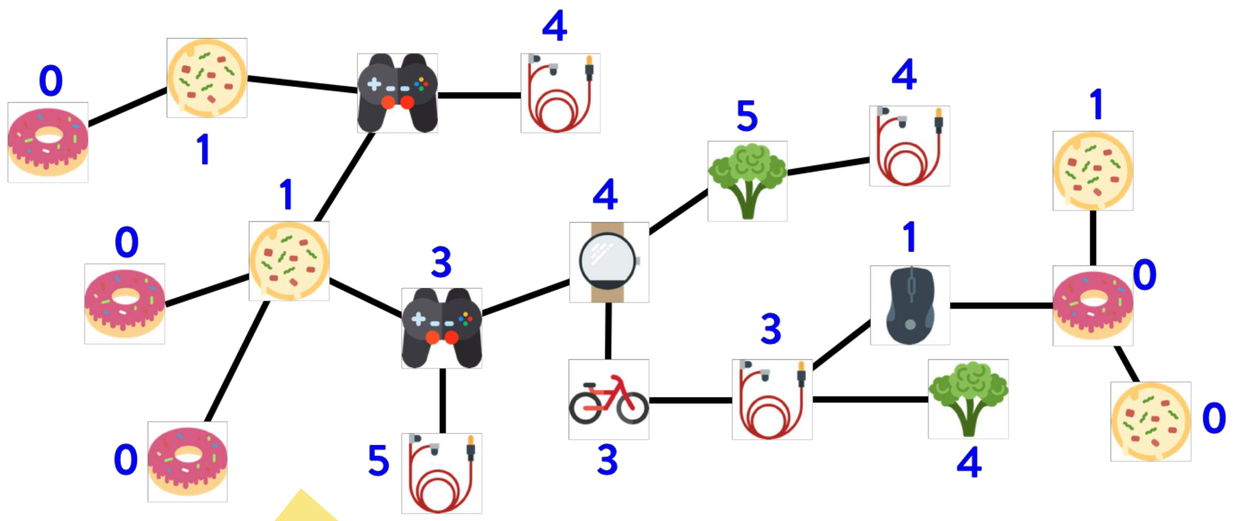
UNIVERSITY OF TRENTO
dbTrento

Giulia Preti[1], Matteo Lissandrini[1], Davide Mottin[2], Yannis Velegrakis[1]
[1]University of Trento [2]Hasso Plattner Institute

HPI

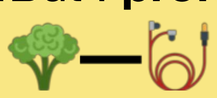
Modern Applications Offer **Personalized** Experiences. One Size Fits None!

Preferences as Weights

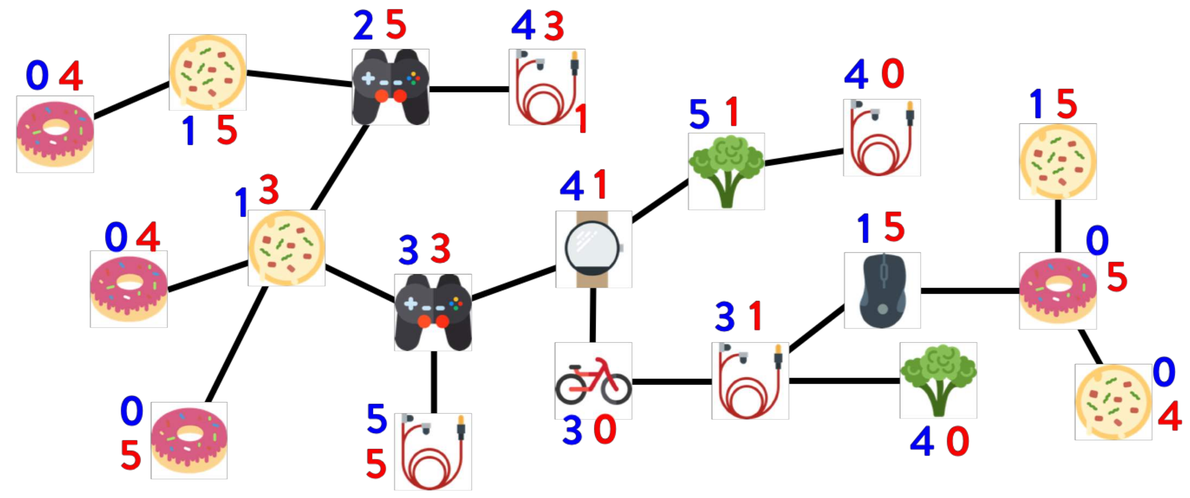


This is **popular**

…But I **prefer**

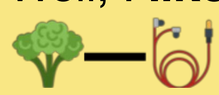


What Happens with Multiple Users?



I **like**

Well, I **like**



Pattern Mining in Graphs

INPUT: a graph, a threshold, and a **scoring function**

GOAL: find the patterns in the graph with score **above the threshold**

In frequent pattern mining the score is the <u>support</u> → **For weighted graphs, the score should depend on the <u>weights</u> too**

Pattern Mining in Multi-weighted Graphs

INPUT: a **multi-weighted** graph, a threshold, and a **scoring function**

GOAL: **for each set** of weights, find the patterns with score **above the threshold**

Each user has her <u>own</u> preferences → **<u>Different</u> weights in the graph!**

MNI-compatible scoring functions:

1. the score monotonically increases with the **weights** of the appearances
2. the score monotonically increases with the **number** of large-weighted appearances
3. the score is **upperbounded** by the MNI support

Our Scoring Functions

«Which isomorphic subgraphs contribute to the pattern score?»

ALL: subgraphs <u>whose edges</u> have large weights
ANY: subgraphs with <u>at least one</u> large edge weight
SUM: subgraphs where the <u>sum</u> of the edge weights is large
AVG: subgraphs with a large <u>average</u> edge weight

Our Exact Solution

Pattern matching as a **Constraint Satisfaction Problem** (X, D, C):

- X contains a variable for each pattern node
- D contains a set of matching graph nodes (candidates) for each variable
- C is a set of constraints enforcing the topology of the pattern

A solution to the CSP corresponds to a **subgraph isomorphic** to the pattern

Computation of the **Pattern Score**

1. For each candidate of each variable in X, search for a valid assignment
2. If no assignment is found, **discard** the candidate
3. Check the weights of the assignments against the scoring function
4. The score is the **min number of marked nodes** per pattern node

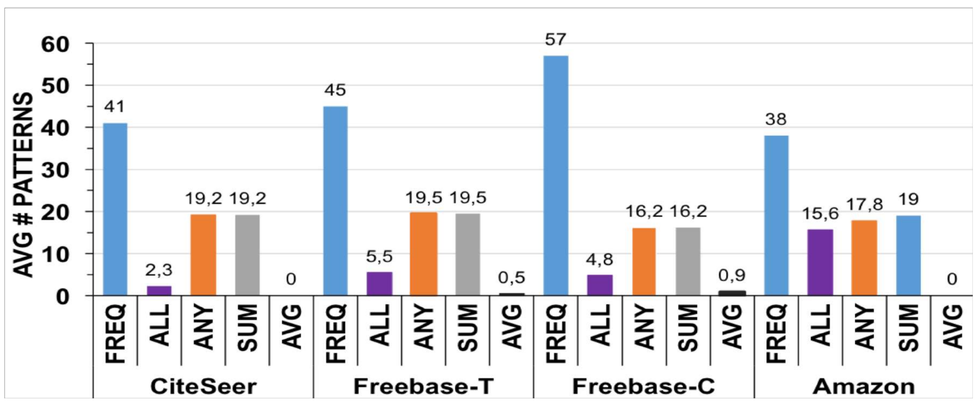
Our Approximate Solution

Trade accuracy for performance by exploiting the **similarities** between the sets of weights

1. Generate k **representative** functions
   - Create the feature vectors
   - Identify the similar functions
   - Generate the maximum-weight vectors
2. Compute k **approximate** sets of patterns

Save Memory! **Spurious** Patterns!

Frequent vs Relevant Patterns

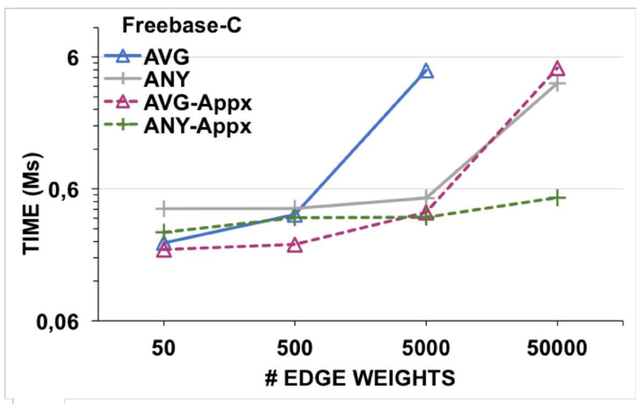


The scoring function affects the patterns returned

Frequent patterns upper bound on relevant patterns

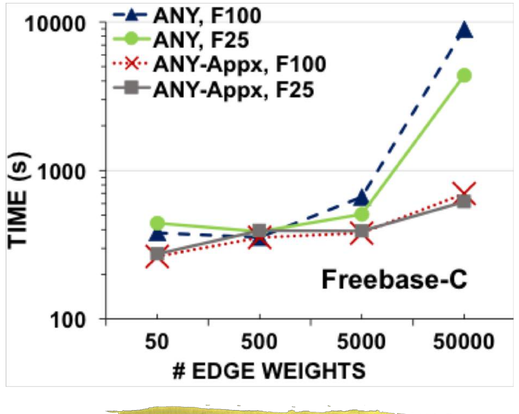
Relevant patterns reduce information overflow

Scalability



Performance not worse than FPM

Impact of Weight Dispersion



Stable performance