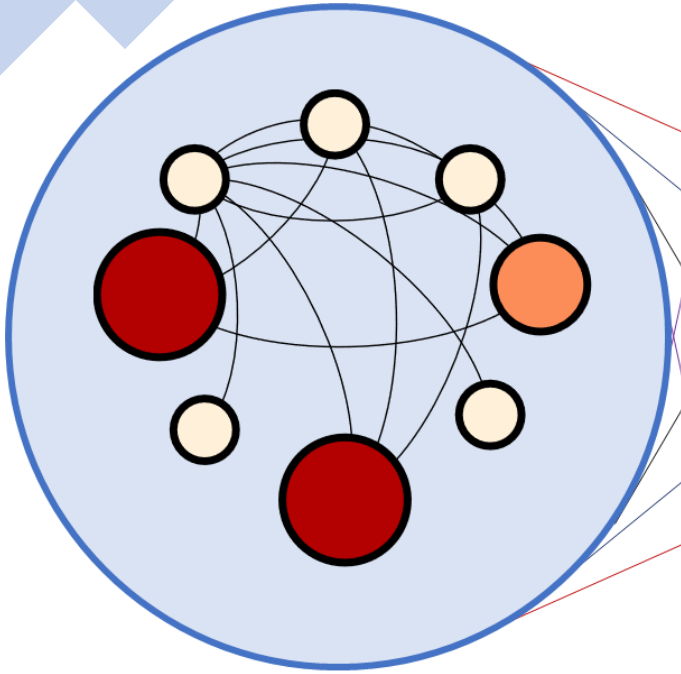


RELISON

REcommending **LI**nks in **SO**cial **N**etworks

Javier Sanz-Cruzado

Glasgow IR Away Day

December 12, 2022

Development

Javier Sanz-Cruzado

University of Glasgow

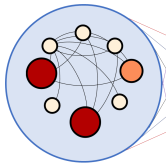
Pablo Castells

UAM
Universidad Autónoma de Madrid

amazon

Outline

1. Introduction to networks
2. What is RELISON?
3. Installation
4. How to use it
5. Future plans

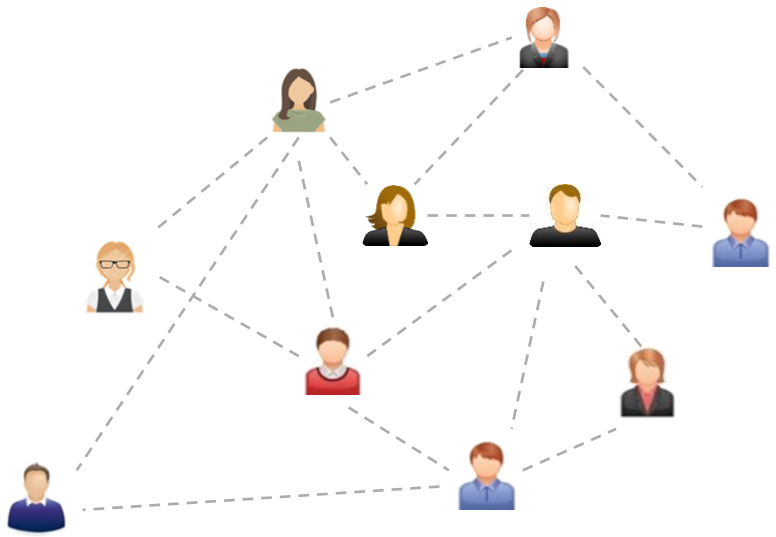


RELISON

REcommending LInks in SOcial Networks

Introduction to networks

The basics – Networks



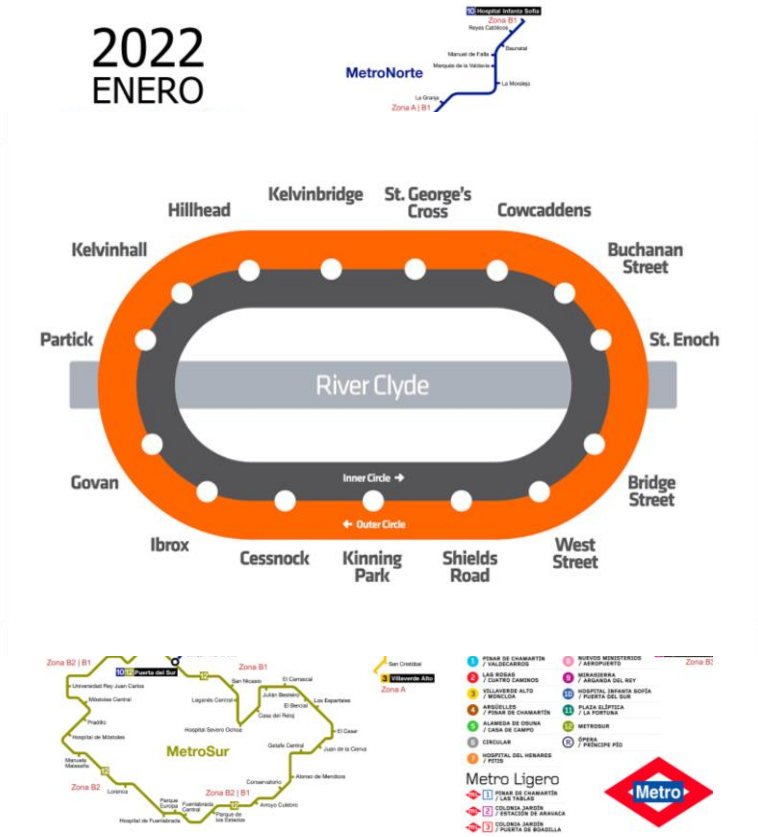
- Two types of objects:
  - Nodes
  - Relations (connections between nodes)
- Mathematically modelled as graphs
- Networks are everywhere!

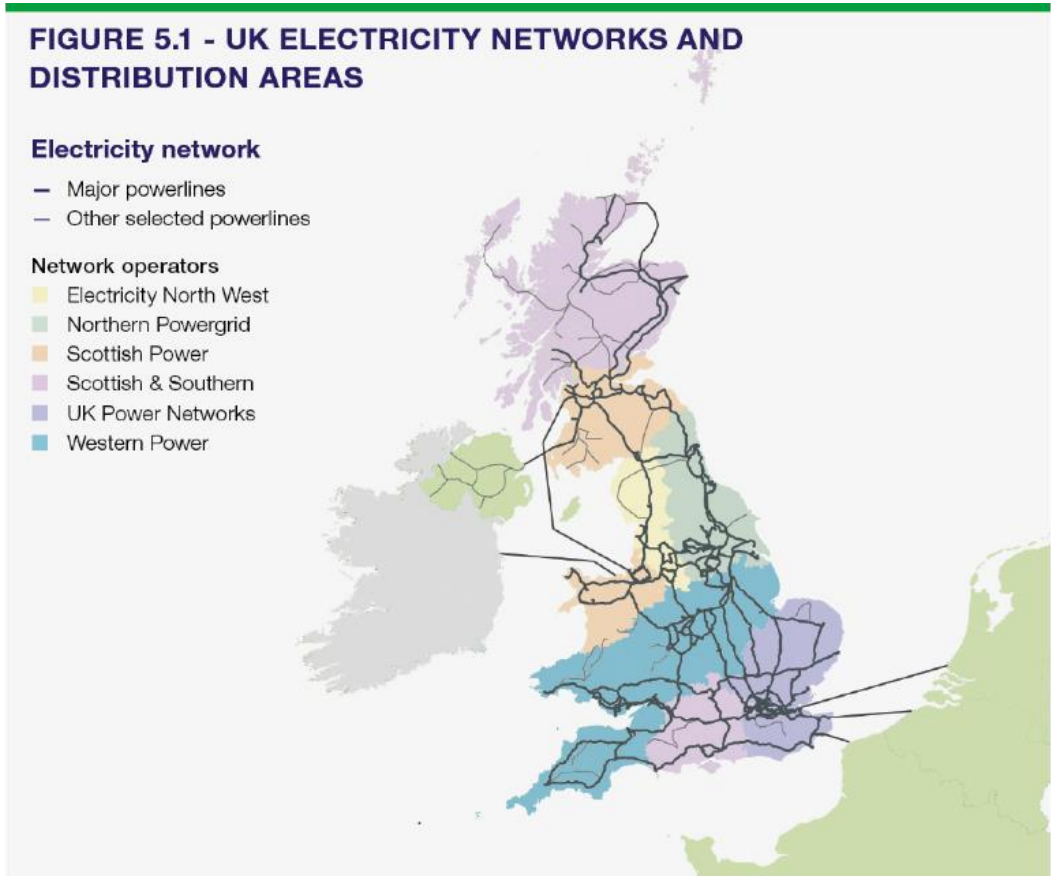
Networks are everywhere!

- Social networks (connecting people)
  - Twitter
  - Facebook
  - LinkedIn
  - Goodreads
  - TikTok
  - …

Networks are everywhere!

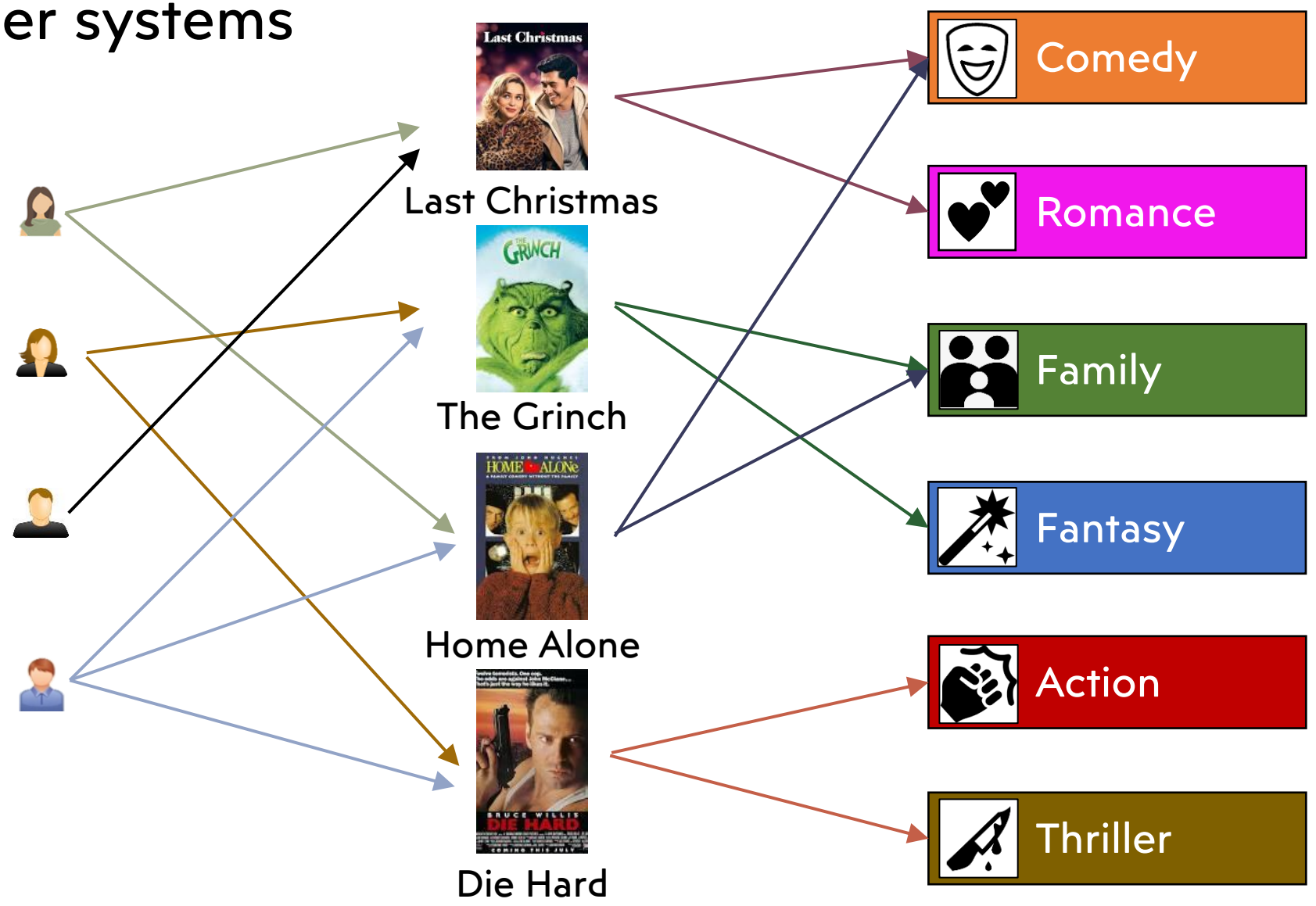
- Transport / energy networks





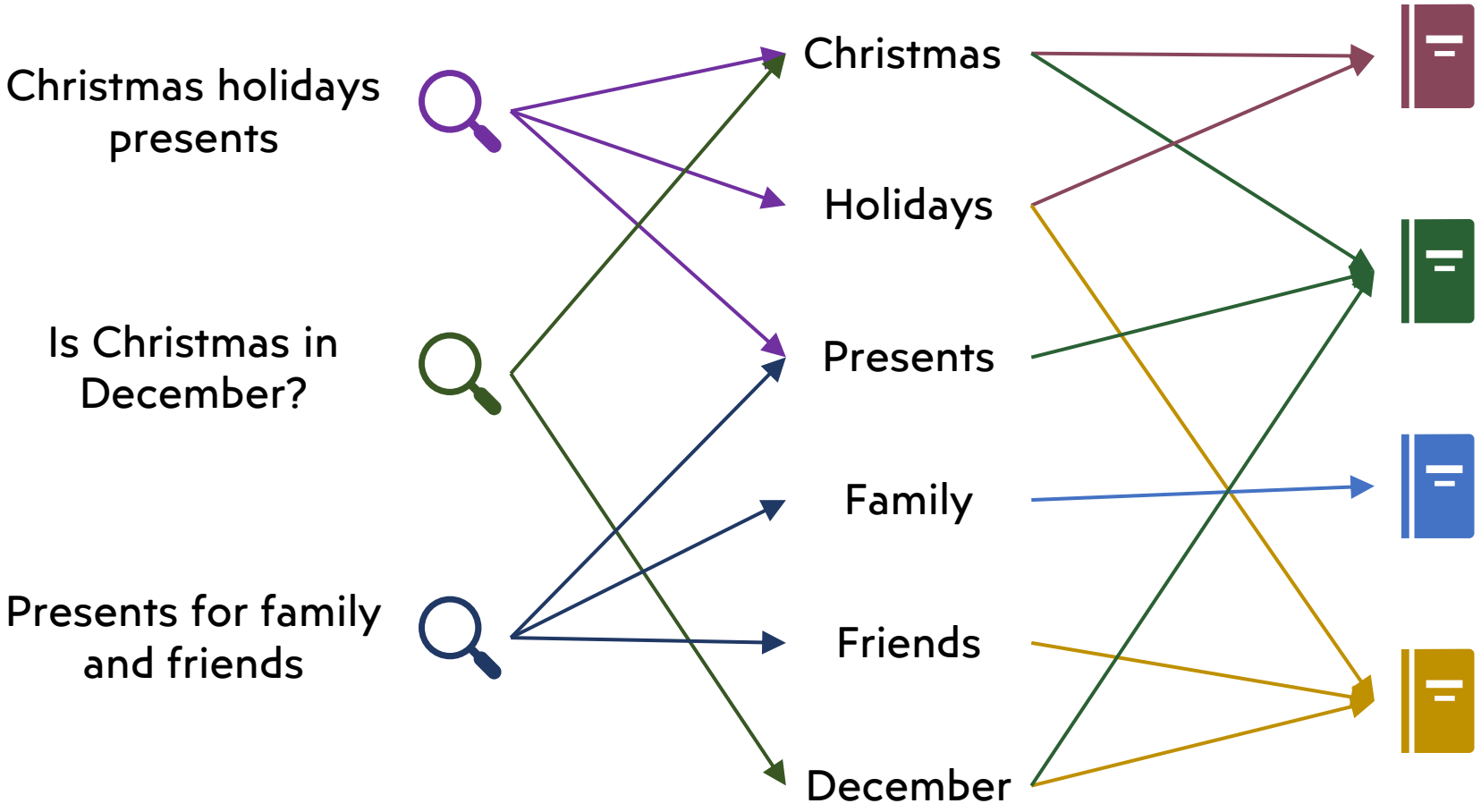
Networks are everywhere!

- Recommender systems



Networks are everywhere!

- Search engine

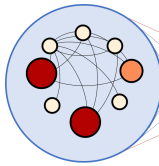


Networks are everywhere!

- In summary, many things can be modelled as a network / graph

- So you might need some tool to:
  - Create and manipulate networks
  - Analyze the structure of those networks
  - Predict future links
  - Etc.

- RELISON fits that gap!



RELISON

REcommending LInks in SOcial Networks

What is RELISON?

What is RELISON?

- **RELISON:** REcommending LInks in SOcial Networks
- Java library

- Focus:
  - Link recommendation
  - Effects over network structure

FAQ

- Is RELISON only useful for link recommendation?

  No! It offers much more! We'll see in the next slide

- Is RELISON only useful for social networks?

  No! You can use it over general networks

- Are you sure it is only available in Java?

  Yes, as of now (version 1.0.0) it is only available in Java

Functionalities provided by RELISON

Network creation

Network structural analysis

Link recomm.

Information diffusion

Network edition

Community detection

Link prediction

RELISON useful links

OFFICIAL WEBSITE

GITHUB

DOCUMENTATION

RELISON

CORE

- Basic definitions
- Graph generators

SNA

- Social network analysis
- Community detection

LINKPRED

- People recommendation
- Link prediction

DIFFUSION

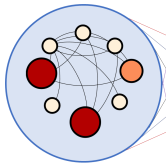
- Simulation of information diffusion dynamics

CONTENT

- Indexing of user-generated contents

EXAMPLES

- Command line programs

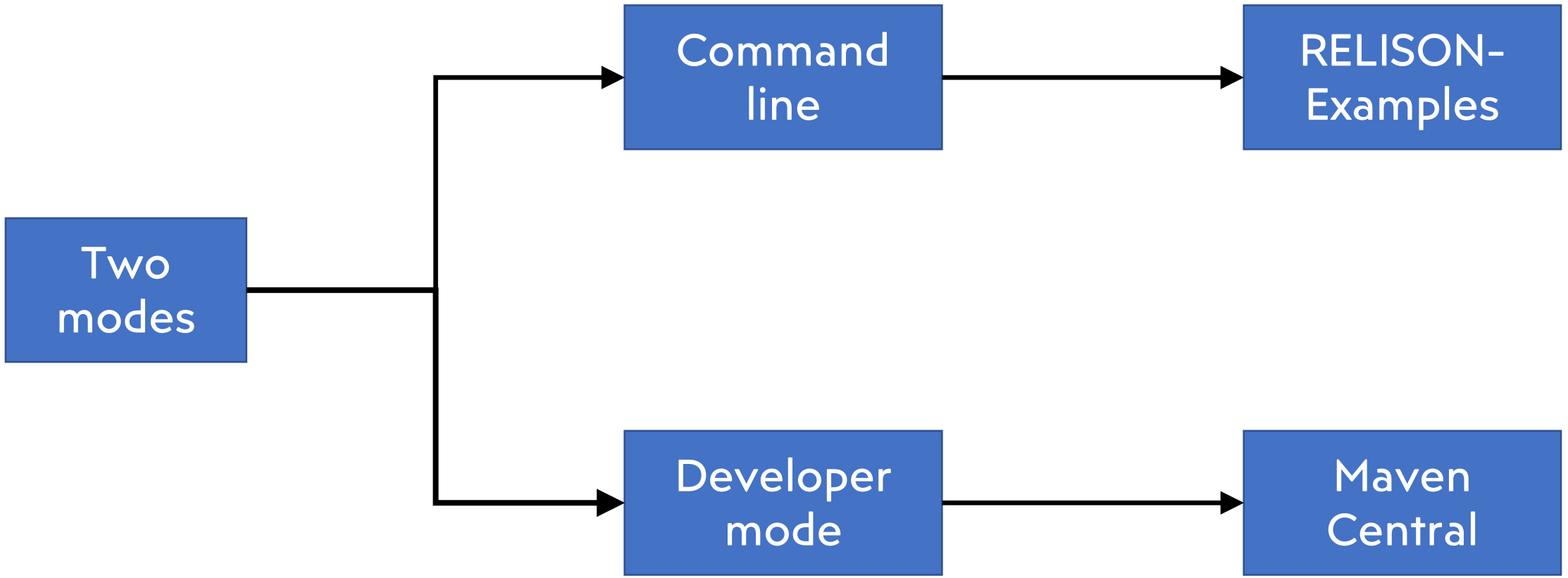


RELISON

REcommending LInks in SOcial Networks

Installation

Installation



Command line

- RELISON provides command line programs to execute multiple functionalities

- Limitations:
  - Graphs must be read from .csv files
  - Node types can only be integers

- Advantage: no need for code

How to use it?

- Step 1: Download the compiled JAR file

```
curl -L https://github.com/ir-uam/RELISON/releases/download/v1.0.0-maven/relison.jar --output relison.jar
```

- Step 2: Execute program

```
java [VM Options] -jar relison.jar [PROGRAM_CODE] [arguments]
```

We'll explore some of the programs during this presentation!

Developer mode

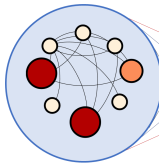
- Used for implementing new methods, metrics, etc.
- **Recommendation:** Maven
  - RELISON is available in Maven Central since last month
  - Add the dependency as:

```
<dependency>
        <groupId>io.github.ir-uam</groupId>
        <artifactId>RELISON-[module-name]<artifactId>
        <version>1.0.0</version>
</dependency>
```

  - Where [module-name] is the name of the module to use

Note about this tutorial

- For time constraints, we will focus here on command line programs
- If you want to use developer mode, refer to the documentation of the library.
- Would you be interested on a presentation like this for “developer mode”?



RELISON

REcommending LInks in SOcial Networks

How to use it

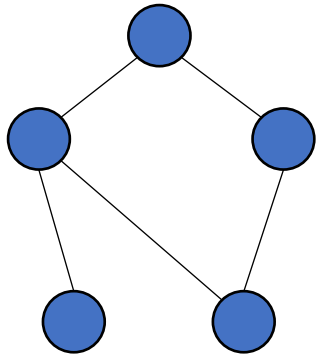
Hands-on

- Do you want to try RELISON as we advance?

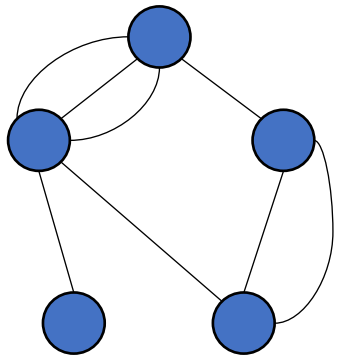
Network creation

Types of graphs supported

Edge number between nodes

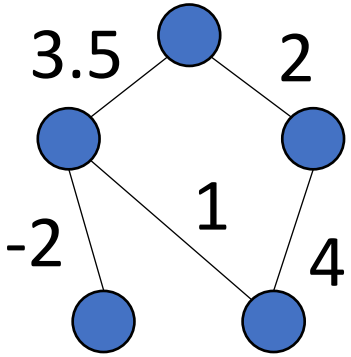


Simple (1)

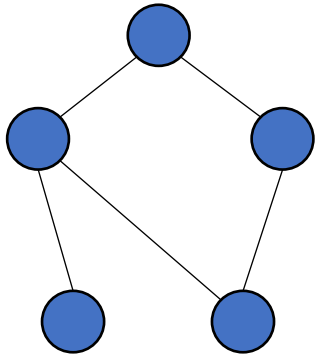


Multigraph (>1)

Edge weights

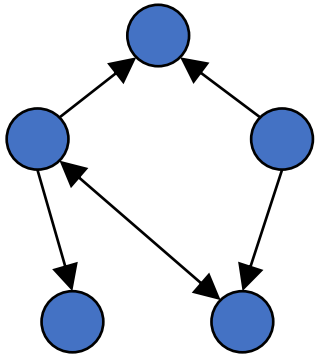


Weighted

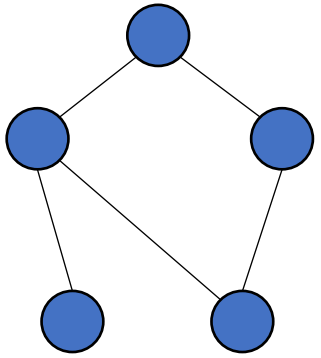


Unweighted

Edge direction



Directed



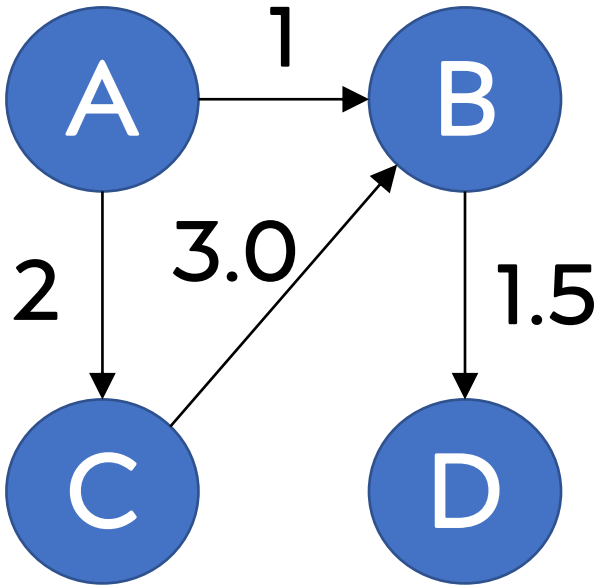
Undirected

Manually crafted networks

- Default format: tab separated .csv

```
Origin \t Destination \t Weight (opt) \t Type (opt)
```

- Example: type less weighted directed network



File:

```
A    B    1.0
A    C    2.0
B    D    1.5
C    B    3.0
```

Automatically created random graphs

- Graph models
  - Define the properties of a network
  - Nodes and edges are created accordingly to a stochastic algorithm.

- In RELISON:
  - Random Erdös networks
  - Preferential attachment
  - Watts-Strogatz
  - Non-stochastic: complete graph, empty graph

Automatically created random graphs (II)

- The *graphgen* program

```
graphgen output_file directed? num_nodes algorithm_name [parameters]
```

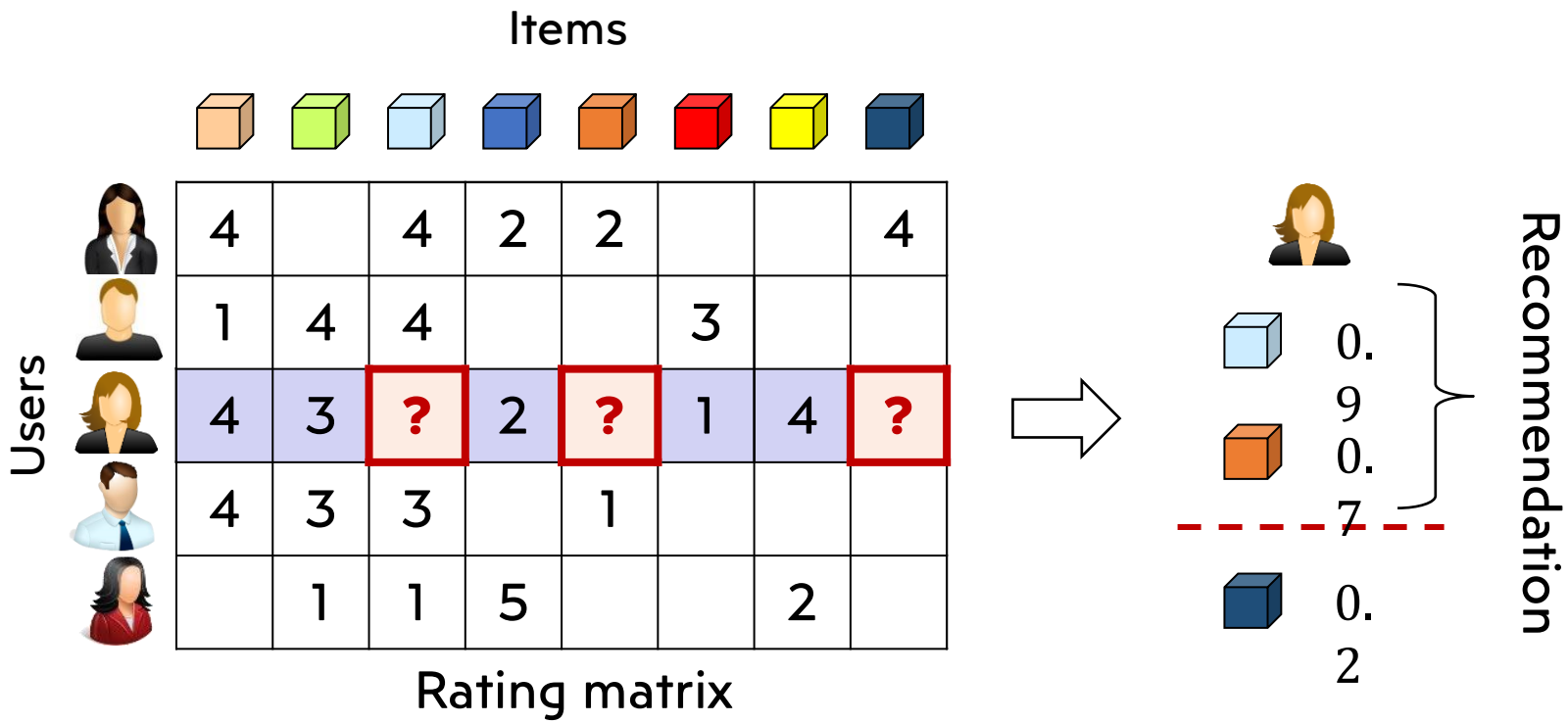
- Input:
  - **output_file:** file on which to store the graph
  - **directed:** true if the graph is directed, false otherwise
  - **num_nodes:** the number of nodes
  - **algorithm_name:** the name of the graph generation algorithm
  - **[parameters]:** the parameters for the algorithm
- Output:
  - A tab separated .csv file with the graph

LINKPRED

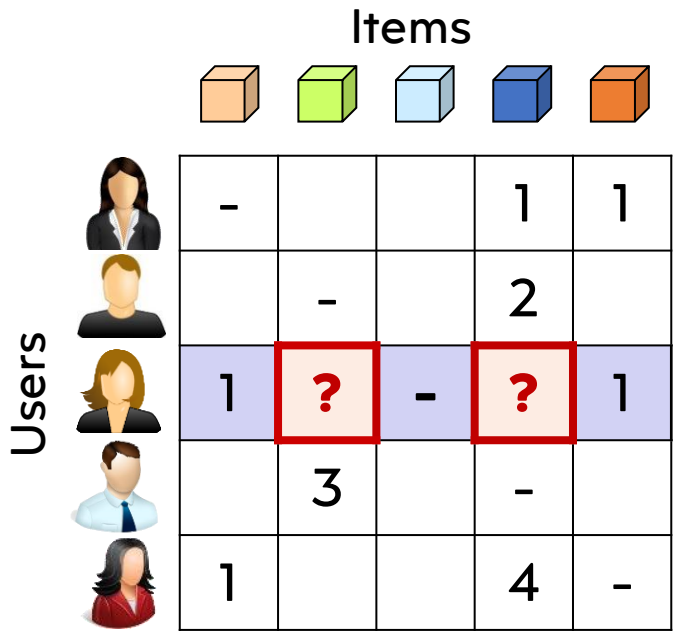
RELISON
REcommending LInks in SOcial Networks

Link recommendation & link prediction

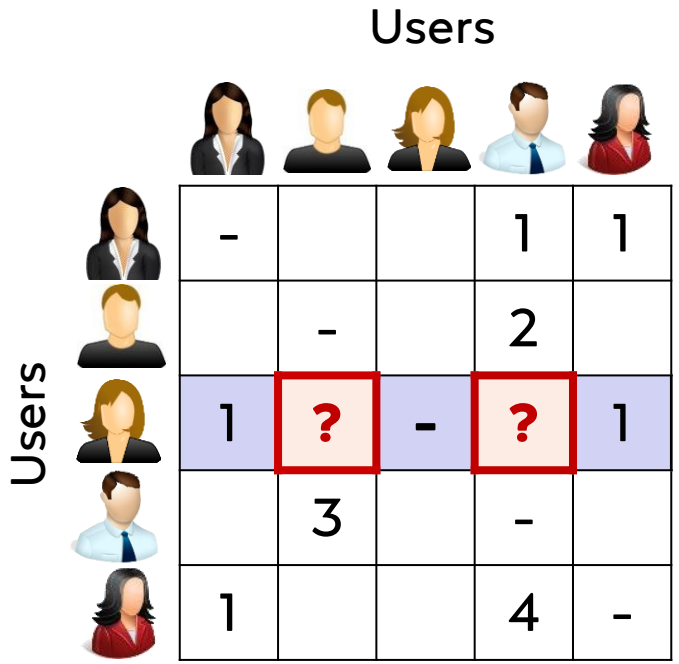
Recommendation task

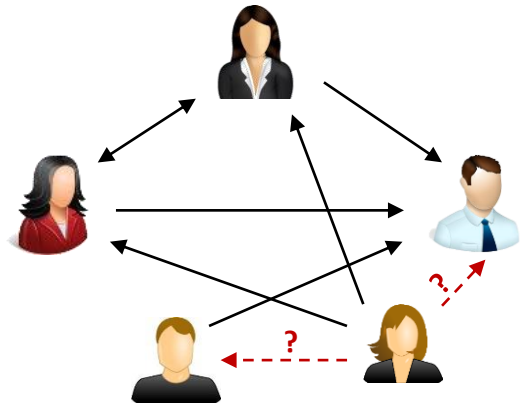


Link / People / Contact recommendation



Link / People / Contact recommendation





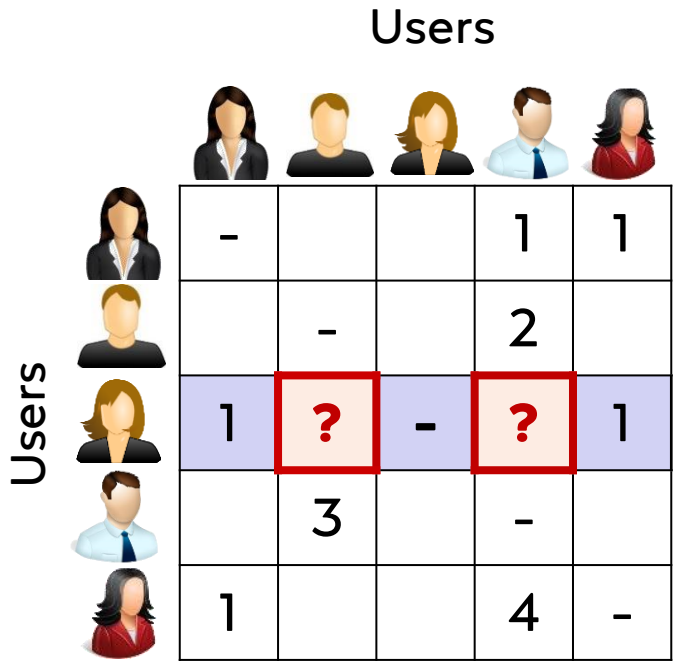
- Items = users
- Availability of social relationships
- Rating matrix = adjacency matrix

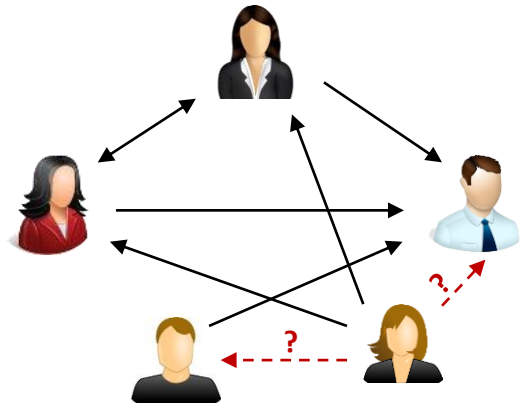
Importance of link recommendation

- Particular characteristics w.r.t. classic recommendation
  - Development of new methods
  - Use of social network analysis

- Creation of new links
  - Main asset of networks
  - Communication channels
  - Source of information
  - Increase engagement of users

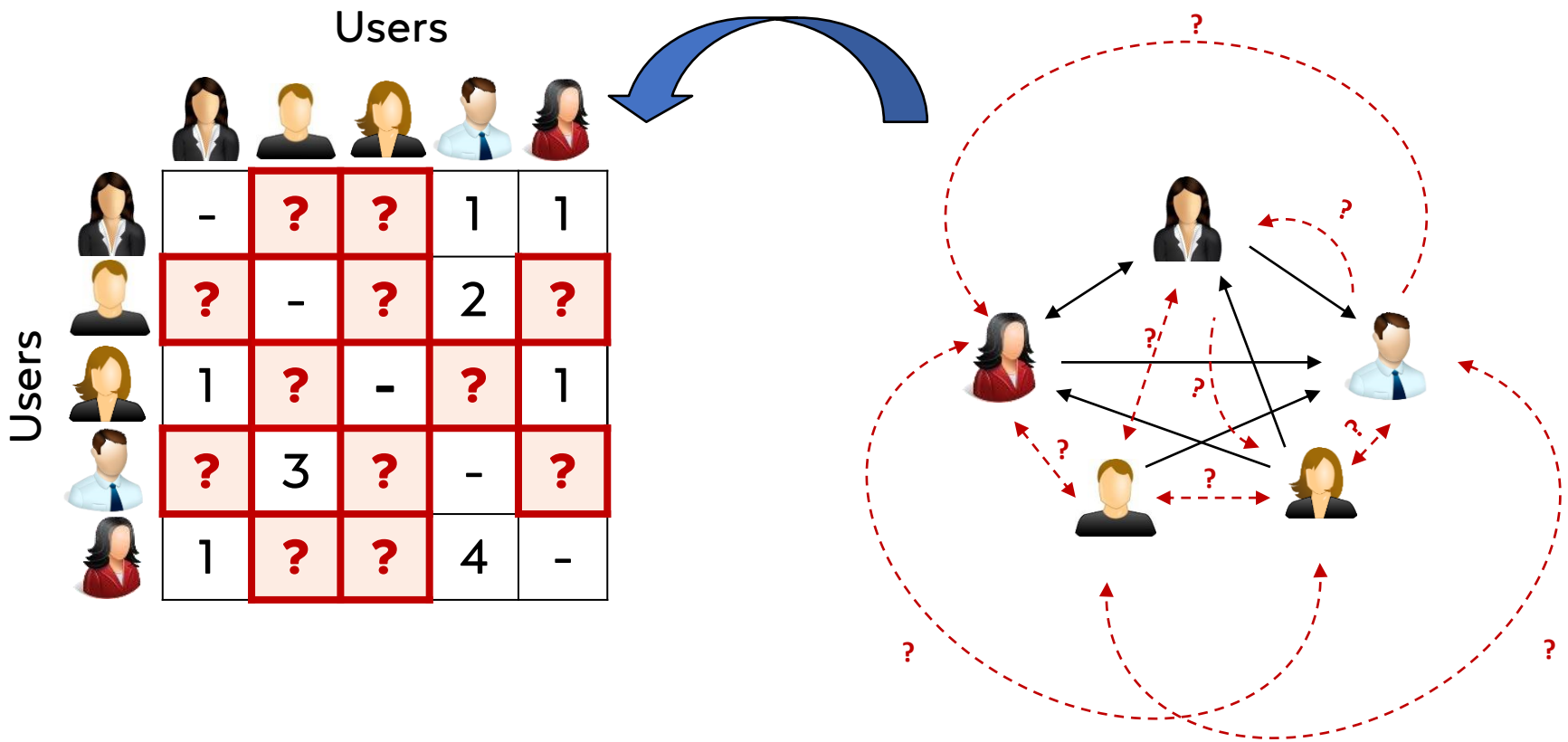
Link / People / Contact recommendation





- Items = users
- Availability of social relationships
- Rating matrix = adjacency matrix

Link prediction



- Which edges will appear in the network in the future?
- Classification problem
- Unique ranking for all possible links

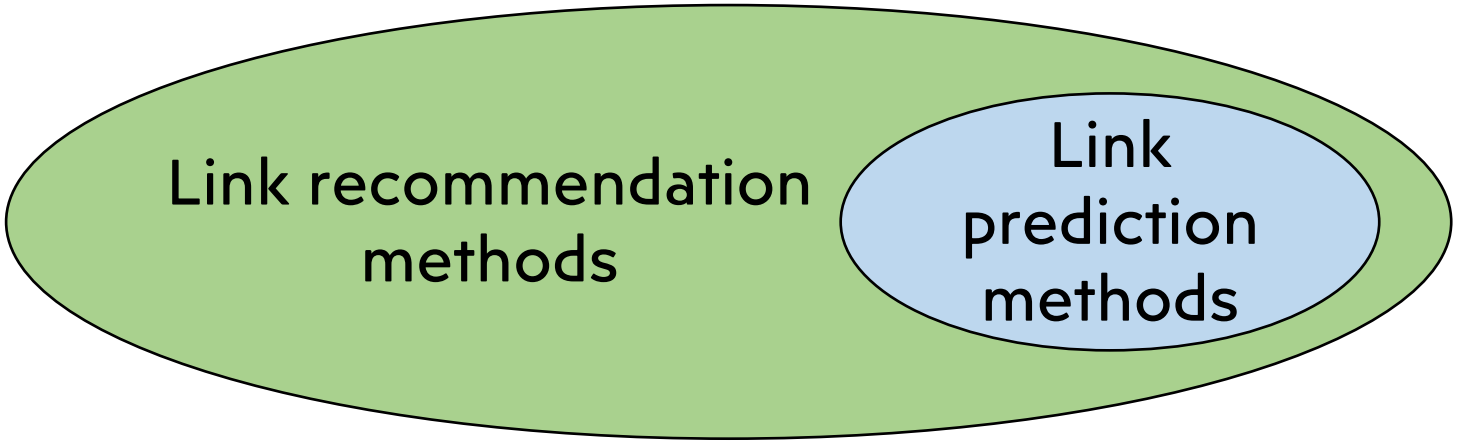
Link recommendation vs. prediction

Link recommendation

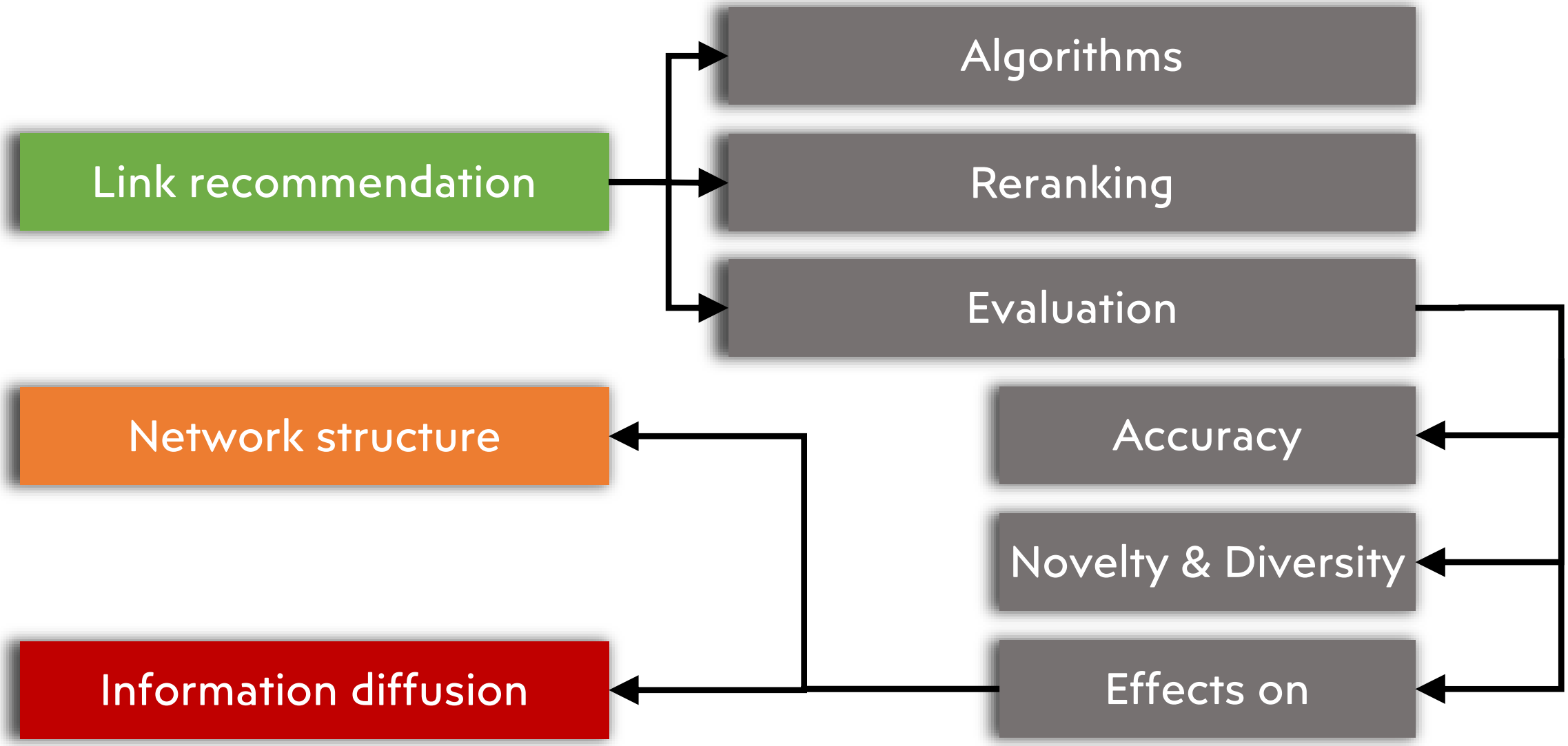
- Social networks (mostly)
- Local ranking problem

Link prediction

- Any network
- Global ranking / classification



Functionalities for link recommendation



Algorithms

Unsupervised		Supervised	
Trivial	• Random • Popularity	Classifiers	• Random forest • Linear regression
Friends of friends	• MCN • IR models	Learning to rank	• LambdaMART
Path based	• Graph distance • Katz similarity		
Random walk	• Personalized PageRank • Money (Twitter)		
Collaborative filtering	• User / Item-based kNN • Implicit MF		
Content based	• Twittomender • Centroid-based CB		

More than 50 algorithms!

Evaluation metrics

- **Accuracy:** IR-based metrics
  - Precision
  - Recall
  - nDCG
  - MAP
- **Diversity:**
  - Intra-list diversity
  - ERR-IA
  - Predicted Gini complement
- **Novelty:**
  - Long tail novelty
  - Unexpectedness
  - Mean prediction distance
- **Effects on networks:** Later, on SNA section

How to use

- **From command line:** the *recommendation* program

```
recommendation train test multigraph directed weighted selfloops
readtypes config output rec-length [optional parameters]
```

  - **train / test:** the training/test networks
  - **multigraph/directed/weighted**: true/false if network is multigraph/directed/weighted
  - **selfloops:** true if we want to read self-loops
  - **readtypes:** true if edges have types and we want to read them
  - **config:** YAML configuration file
  - **output:** output directory in which to store the files
  - **rec-length:** cutoff of the recommendation

Program configuration (YAML)

```
algorithms:
    algorithm name:
        param_name:
            type: int/double/boolean/string/long/orientation
            values: [value1,…,valueN] or value
            range:
            - start: start_val
              end: end_val
              step: step_val
metrics:
    metric_name:
        param_name:
            …
```

Program output

- The recommendation program provides two classes of outputs:

  - **Recommendations:** the generated recommendations. Csv files with the format:

```
Target_user \t Candidate_user \t Score
```

  - **Metrics:** the metrics for the recommendations. A .csv file with the format

```
Variant \t Fraction \t Metric1 \t Metric2 \t … \t MetricN
```

Example configuration (YAML)

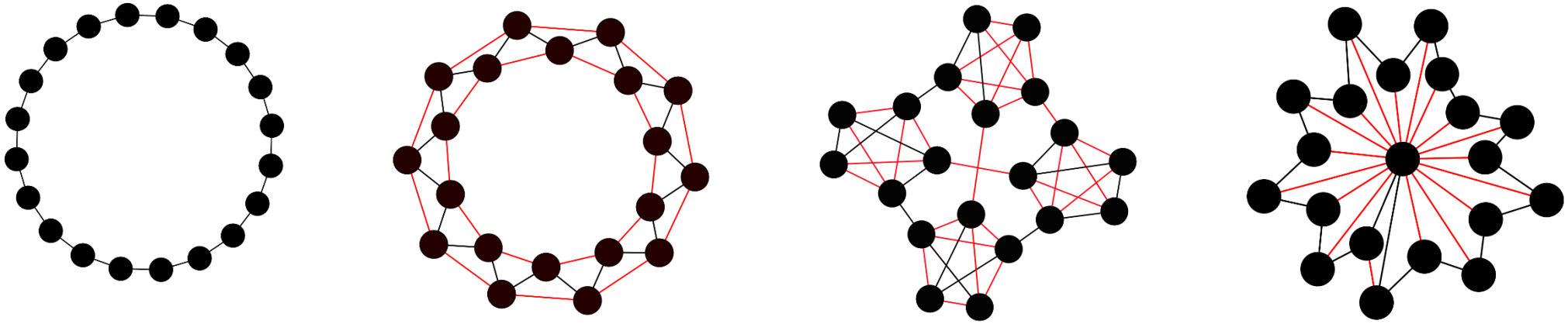
```yaml
algorithms:
    iMF:
        k:
            type: int
            range:
            - start: 10
              end: 300
              step: 10
        alpha:
            type: double
            values: [10.0, 40.0]
        lambda:
            type: double
            values: 150.0
```

```yaml
metrics:
    nDCG:
        cutoff:
            type: int
            values: 10
    Predicted Gini complement:
        cutoff:
            type: int
            values: [1,5,10]
```

Network structural analysis

Studying the structure of networks

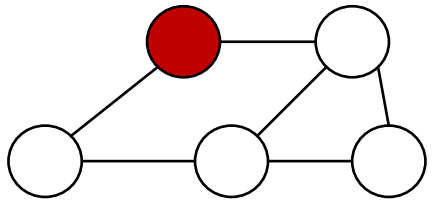
- Different networks present different structures



- Understanding our networks might provide insights on our algorithms
- Many properties can be measured

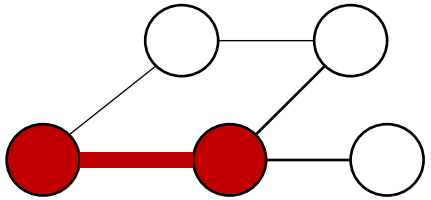
Types of properties

Node



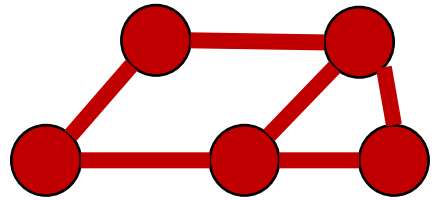
- Degree
- PageRank
- Closeness

Edge/pair



- Distance
- Neighbor overlap
- Betweenness

Graph

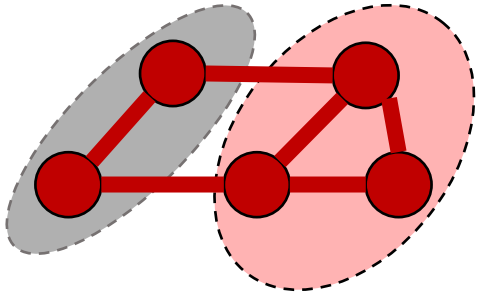


- Clust. coefficient
- Density
- Diameter

Types of properties (community based)

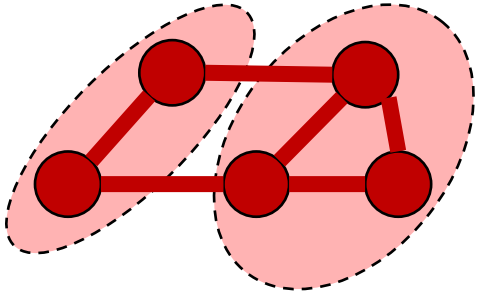
- For now, communities are just partitions of nodes (we'll provide a better definition later)

Individual community



- Degree
- Size
- Volume

Graph (community-based)



- Modularity
- Edge Gini complement

How to use

- **From command line:** the *sna* program
- Computes structural metrics of a network

```
sna network multigraph directed weighted selfloops config output
(-communities comm1,…,commN --distances)
```

  - **network:** the network to analyze
  - **multigraph/directed/weighted**: true/false if network is multigraph/directed/weighted
  - **selfloops:** true if we want to read self-loops
  - **config:** YAML configuration file
  - **output:** output directory in which to store the files
  - **communities:** community files (see later)
  - **--distances:** a flag to pre-compute distances between nodes

Program configuration (YAML)

```yaml
metrics:
    metric name:
        type: vertex/edge/pair/graph/indiv. community/global community
        params:
            param_name:
                type: int/double/boolean/string/long/orientation
                values: [value1,…,valueN] or value
                range:
                - start: start_val
                  end: end_val
                  step: step_val
```

YAML example

```yaml
metrics:
    Clustering coefficient:
        type: graph
        params:
            uSel:
                type: orientation
                values: IN
            vSel:
                type: orientation
                values: OUT
    Eccentricity:
        type: vertex
    Embeddedness:
        type: edge
        params:
            uSel:
                type: orientation
                values: IN
            vSel:
                type: orientation
                values: OUT
```

Program output

- The recommendation program provides many outputs:
- global.txt: a .csv file containing
  - Global metrics
  - Averaged vertex/node/indiv. community metrics
  - Format:

```
Metric \t Value
```

- Directories for every other metric, containing a file per metric
  - For vertex metrics: `Node \t Value`
  - For pair metrics: `Origin \t Destination \t Value`
  - For indiv. community metrics: `Community \t Value`

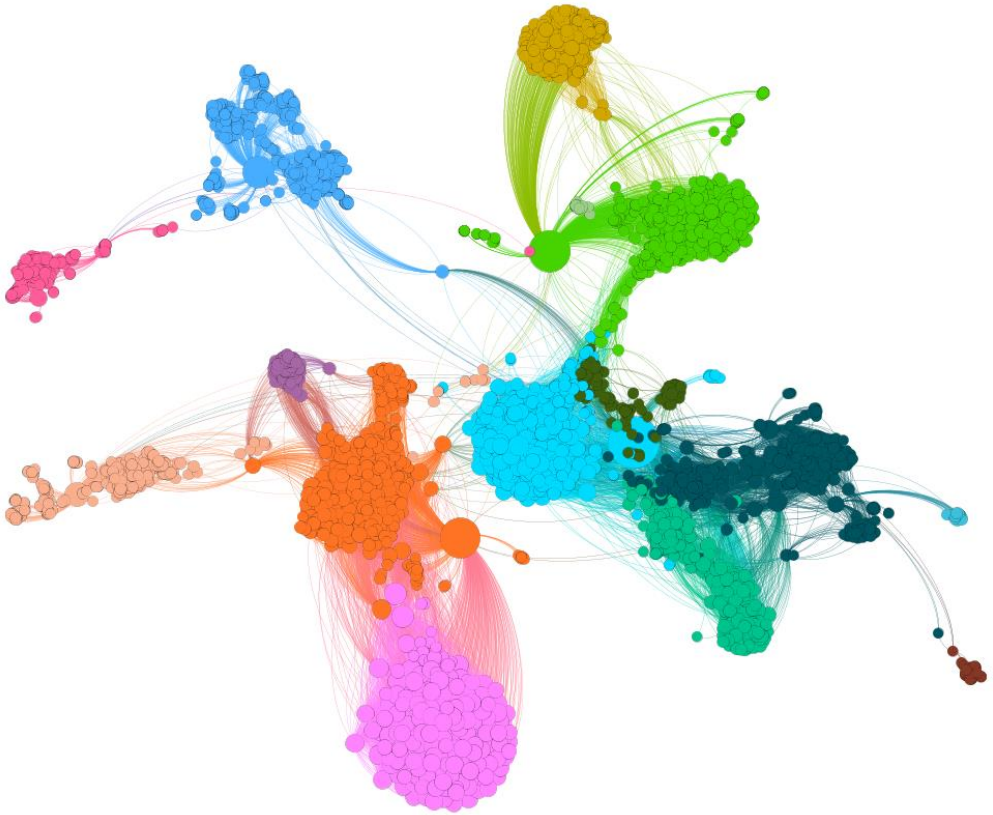
Effects of recommendations

- There is also a program for measuring the effect of recommendations
- The program is named ***effects***
- Configuration file is the same
- It only computes global properties
- More information in the documentation of the library

Community detection

Communities

- **Homophily:** similar nodes tend to relate to each other
- It is a characteristic of real world networks
  - Groups of tightly connected nodes
  - Barely connected to each other
- We name those clusters of nodes **communities**



Community detection algorithms

- **Connectedness-based approaches**
  - How many nodes can we reach from a particular node?
  - In undirected networks: connected components
  - In directed networks: strongly / weakly connected components

- **Modularity-based approaches**
  - Modularity measures how good a clustering is
  - Compares:
    - Links inside communities
    - Links between communities we would have in a random graph keeping the degree distribution
  - Methods: Louvain, Infomap

How to use

- **From command line:** the *communities* program

```
communities graph multigraph directed weighted self_loops config
output_dir
```

  - **graph:** the network on which we want to detect communities
  - **multigraph/directed/weighted**: true/false if network is multigraph/directed/weighted
  - **selfloops:** true if we want to read self-loops
  - **config:** YAML configuration file
  - **output:** output directory in which to store the files

Program configuration (YAML)

```
algorithms:
    algorithm name:
        param_name:
            type: int/double/boolean/string/long/orientation
            value: value
```

Example:

```
algorithms:
    Louvain:
        threshold:
            type: double
            value: 0.0001
```

Program output

- The recommendation program provides an output for each selected algorithm
- It includes the community partition
- CSV file (tab separated)
- Format

```
Node \t Community
```

Information diffusion

Information diffusion

- Edges in networks (and in special social networks) are channels for communicating information

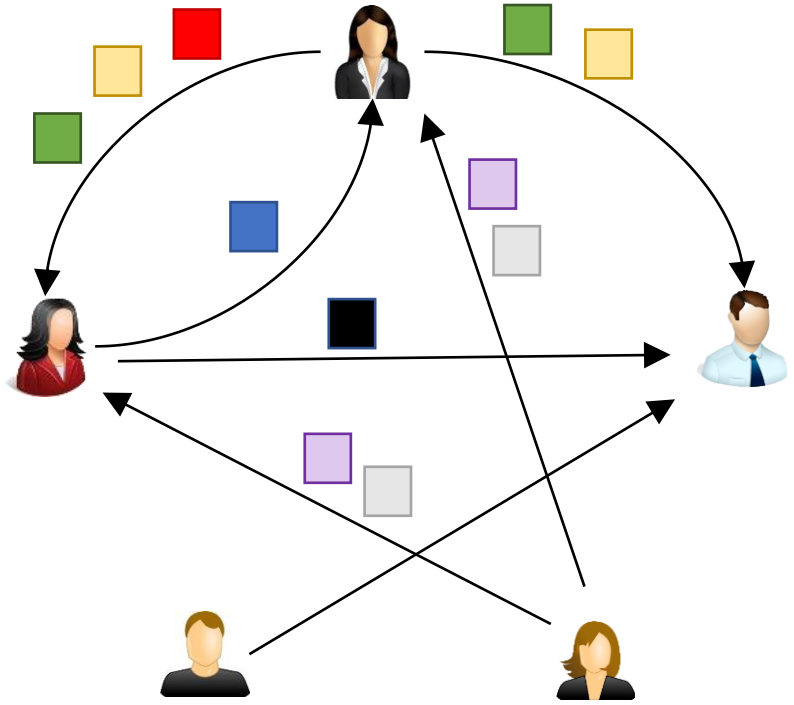
- Example: We read tweets thanks to our followed users

- The information diffusion process is complex
- Mechanisms of it are not exactly known
- Some models have been proposed

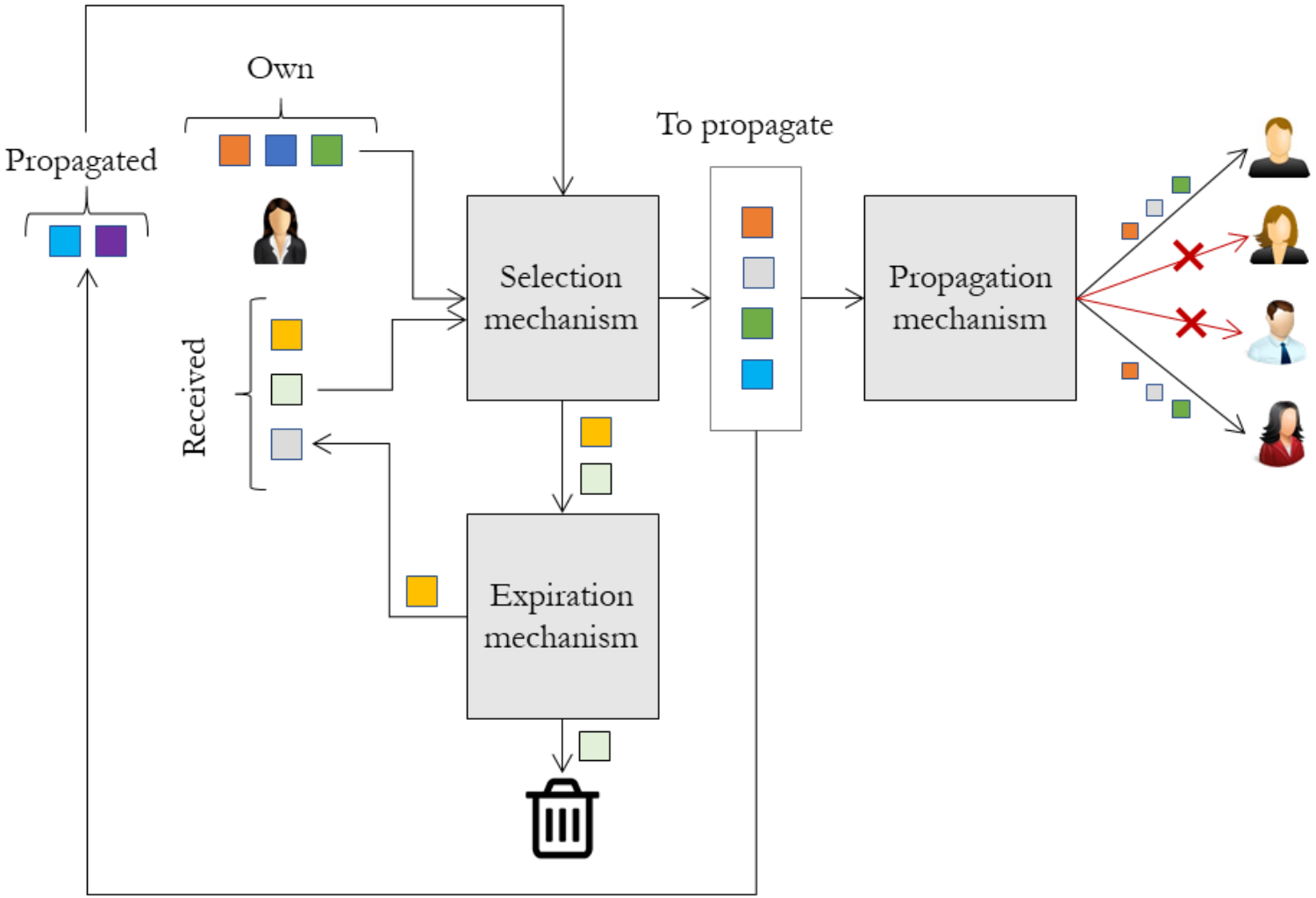
Information diffusion in RELISON

- Based on simulation
- Simulate the simultaneous diffusion of different pieces of information

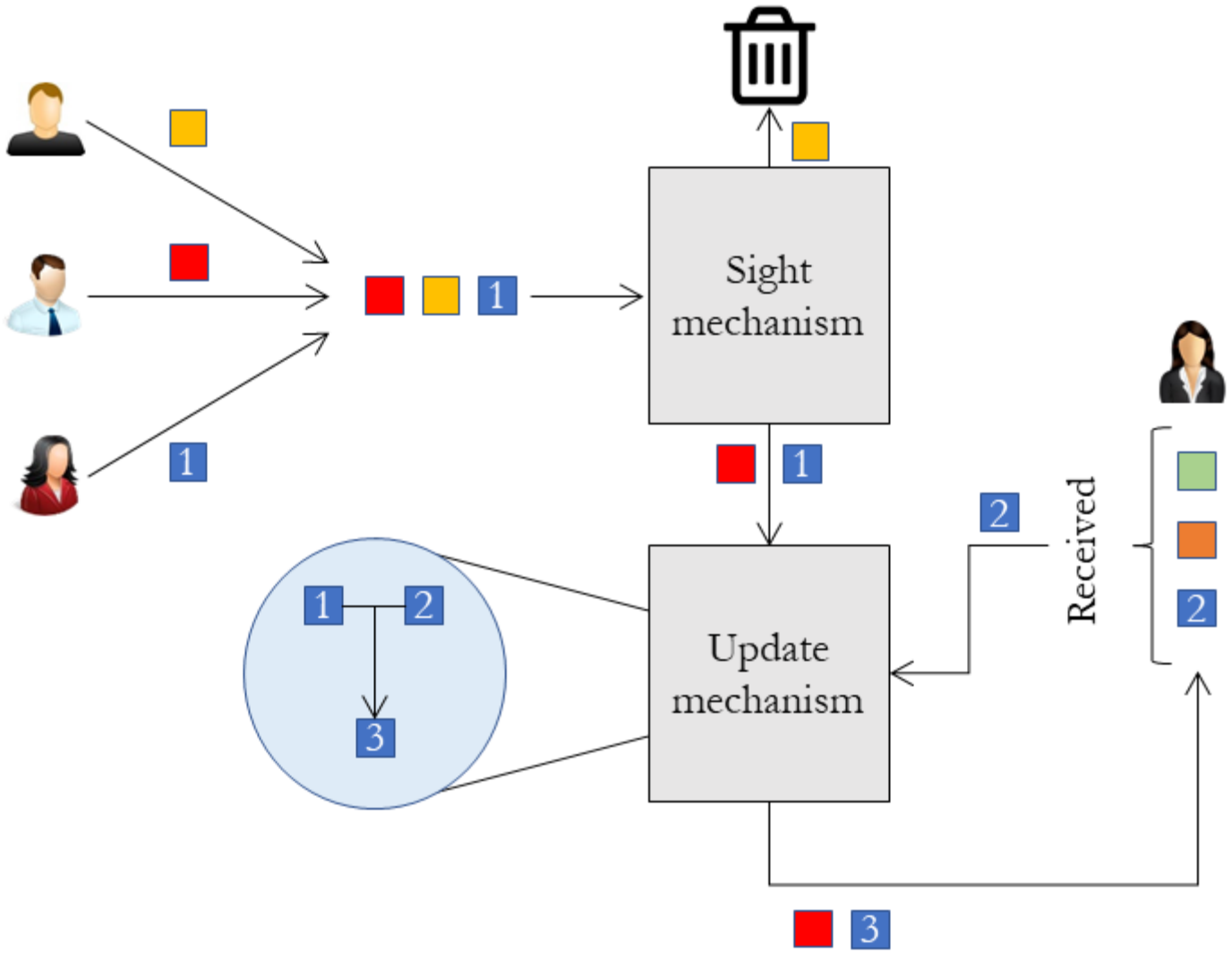
- Highly configurable simulation
- Multiple components

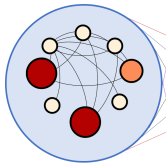


Information spread



Information reception





RELISON

REcommending LInks in SOcial Networks

Future directions

Future directions

- **Knowledge graph support**
- Node / graph embeddings
- Python wrapper (pyRELISON)
- Network visualization
- Suggestions?

Do you want to help?

- Help to improve RELISON wanted!

- Contact me!
  - Office: F111
  - E-mail: javier.sanz-cruzadopuig@glasgow.ac.uk
  - Skype
  - Teams
  - Twitter….

- Together we can make RELISON better!

Do you want to know more?

OFFICIAL WEBSITE

GITHUB

TWITTER