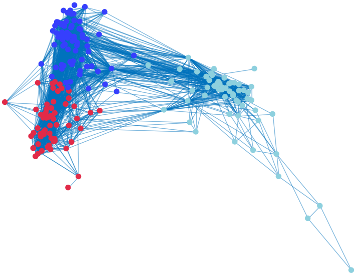
# Networks and Algorithms



Barbara Ikica

Computational Social Science seminar

29 October 2019

# Main ingredients

**Algorithm**

*"A set of mathematical instructions or rules that, especially if given to a computer, will help to calculate an answer to a problem."*

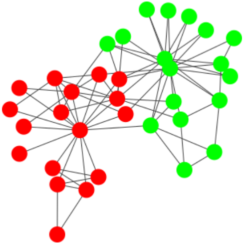
*(Cambridge Dictionary)*

# Main ingredients

**Algorithm**

*"A set of mathematical instructions or rules that, especially if given to a computer, will help to calculate an answer to a problem."*

*(Cambridge Dictionary)*

**Network**

*"A network is, in its simplest form, a collection of points joined together in pairs by lines."*

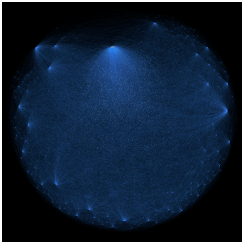
*(Newman, Networks: An Introduction, 2010)*

# Main ingredients

**Algorithm**

*"A set of mathematical instructions or rules that, especially if given to a computer, will help to calculate an answer to a problem."*

*(Cambridge Dictionary)*

**Network**

*"A network is, in its simplest form, a collection of points joined together in pairs by lines."*

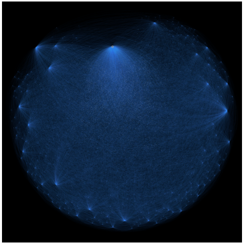
*(Newman, Networks: An Introduction, 2010)*

# Main ingredients

**Algorithm**

*“A set of mathematical instructions or rules that, especially if given to a computer, will help to calculate an answer to a* ***problem****.”*

*(Cambridge Dictionary)*

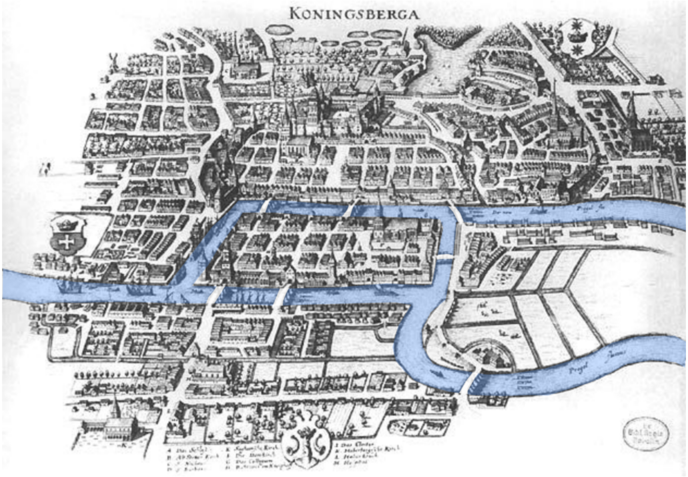



**Network**

*“A network is, in its simplest form, a collection of points joined together in pairs by lines.”*

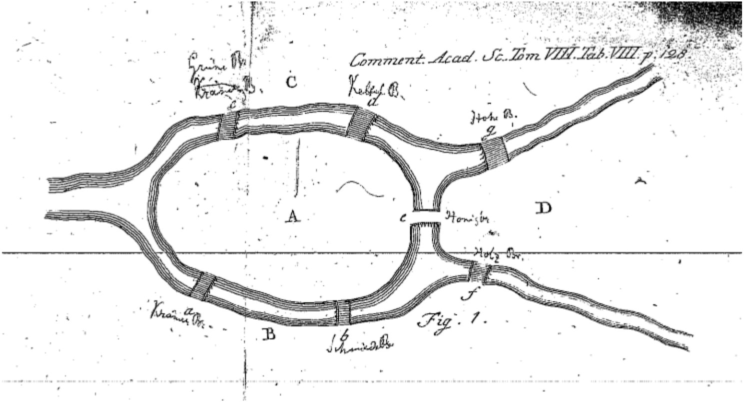
*(Newman, Networks: An Introduction, 2010)*

# Königsberg bridge problem



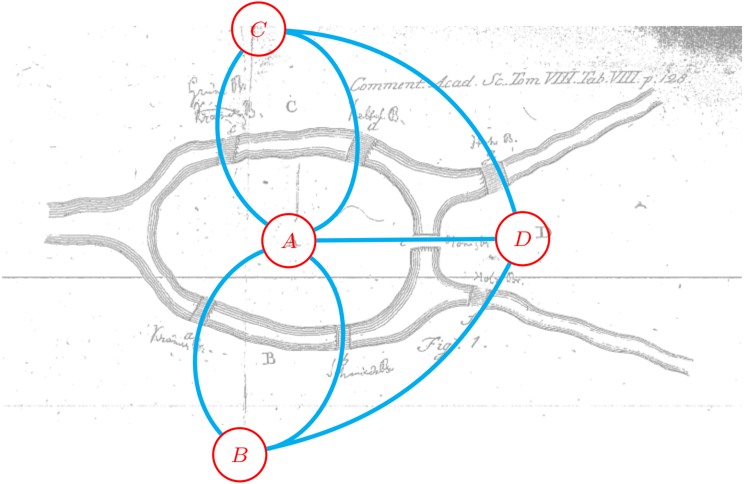
# Königsberg bridge problem

*Does there exist a route that crosses each of the seven bridges exactly once?*

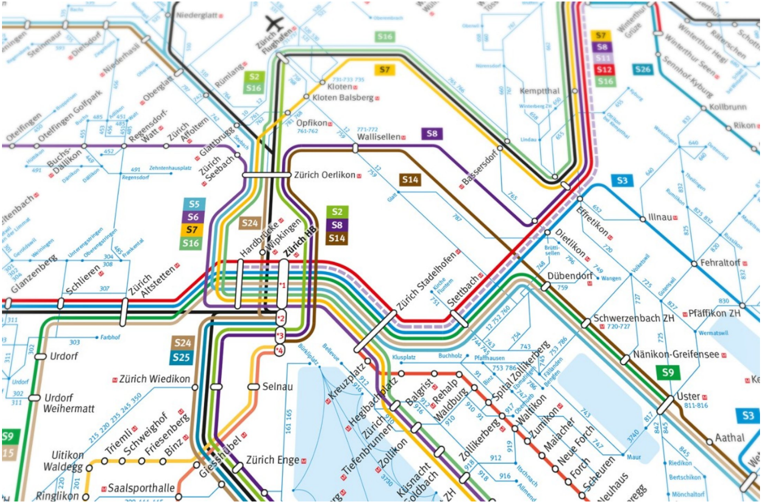


# Königsberg bridge problem

*Does there exist a route that crosses each of the seven bridges exactly once?*



# Meanwhile, in Zürich ...



# Outline

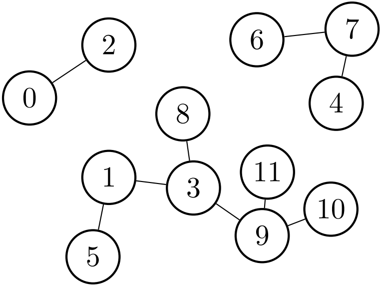
- **Network algorithms**
  - data representation
  - computational complexity

# Outline

- **Network algorithms**
  - data representation
  - computational complexity

- **Examples**
  - Centrality indices – PageRank
  - Community detection – the modified Petford–Welsh algorithm

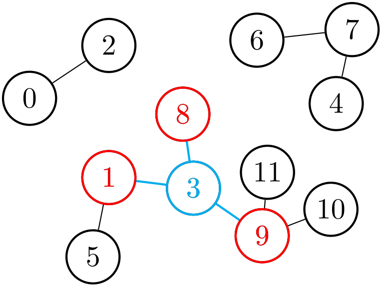
# Adjacency matrix




$$A = \begin{bmatrix} 0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&1&1&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0 \end{bmatrix}$$

$$\boxed{A_{ij} = \begin{cases} 1; & ij \in E, \\ 0; & \text{otherwise.} \end{cases}}$$

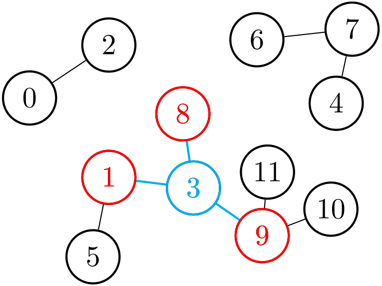
# Adjacency matrix




$$A = \begin{bmatrix} 0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&1&1&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0 \end{bmatrix}$$

$$\boxed{A_{ij} = \begin{cases} 1; & ij \in E, \\ 0; & \text{otherwise.} \end{cases}}$$

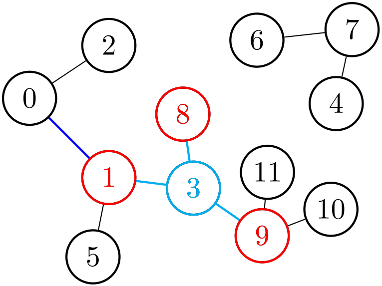
# Adjacency matrix




$$A = \begin{bmatrix} 0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&1&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&1&1&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \end{bmatrix}$$

$$\boxed{A_{ij} = \begin{cases} 1; & ij \in E, \\ 0; & \text{otherwise.} \end{cases}}$$

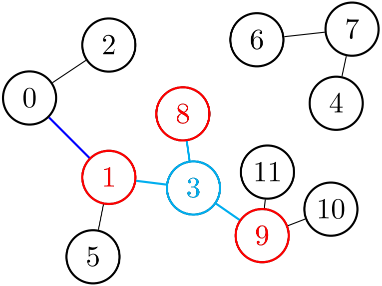
# Adjacency matrix




$$A = \begin{bmatrix} 0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&1&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&1&1&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \end{bmatrix}$$

$$\boxed{A_{ij} = \begin{cases} 1; & ij \in E, \\ 0; & \text{otherwise.} \end{cases}}$$

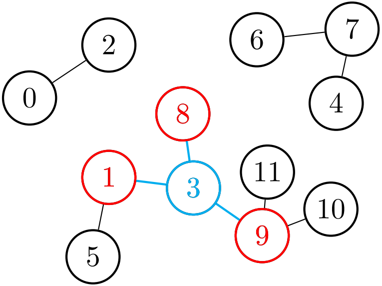
# Adjacency matrix




$$A = \begin{bmatrix} 0&1&1&0&0&0&0&0&0&0&0&0\\ 1&0&0&1&0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&1&1&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0 \end{bmatrix}$$

$$\boxed{A_{ij} = \begin{cases} 1; & ij \in E, \\ 0; & \text{otherwise.} \end{cases}}$$

# Adjacency matrix




$$A = \begin{bmatrix}
0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&1&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&1&0&1&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0
\end{bmatrix}$$

$$\boxed{A_{ij} = \begin{cases} 1; & ij \in E, \\ 0; & \text{otherwise.} \end{cases}}$$

# Adjacency list

$$A = \begin{bmatrix} 0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&1&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&1&1&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \end{bmatrix}$$

# Adjacency list

$$
A \;=\; \begin{bmatrix}
0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&1&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&1&0&1&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0
\end{bmatrix}
\;\longrightarrow\;
\begin{bmatrix}
[2]\\
[3,5]\\
[0]\\
[1,8,9]\\
[7]\\
[1]\\
[7]\\
[4,6]\\
[3]\\
[3,10,11]\\
[9]\\
[9]
\end{bmatrix}
$$

# Adjacency list

$$A = \begin{bmatrix} 0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&1&1&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0 \end{bmatrix} \longrightarrow \begin{bmatrix} [2]\\ [3,5]\\ [0]\\ [1,8,9]\\ [7]\\ [1]\\ [7]\\ [4,6]\\ [3]\\ [3,10,11]\\ [9]\\ [9] \end{bmatrix}$$

# Adjacency list

$$A = \begin{bmatrix} 0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&1&1&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0 \end{bmatrix} \longrightarrow \begin{bmatrix} [2]\\ [3,5]\\ [0]\\ [1,8,9]\\ [7]\\ [1]\\ [7]\\ [4,6]\\ [3]\\ [3,10,11]\\ [9]\\ [9] \end{bmatrix}$$

# Alternatives

- **Adjacency trees** (quick performance on average)
- **Edge lists** (compact representation)
- **Binary heaps** (efficient storage of values/weights)

# Complexity

**Computational complexity**

Computational resources (time, space, memory) needed to run an algorithm.

# Complexity

## Computational complexity

Computational resources (time, space, memory) needed to run an algorithm.

## Time complexity

An estimate how the running time scales with the input.

# Complexity

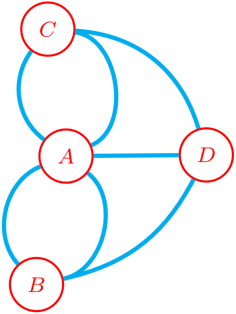
## Computational complexity

Computational resources (time, space, memory) needed to run an algorithm.

## Time complexity

An estimate how the running time scales with the input.

Example: finding the highest degree




| Vertex degrees: | 5 | 3 | 3 | 3 |
|---|---|---|---|---|

Current highest degree: 0

# Complexity

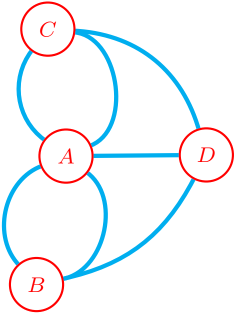
## Computational complexity

Computational resources (time, space, memory) needed to run an algorithm.

## Time complexity

An estimate how the running time scales with the input.

Example: finding the highest degree




Vertex degrees:

| | | | |
|---|---|---|---|
| 5 | 3 | 3 | 3 |

Current highest degree: **5**

# Complexity

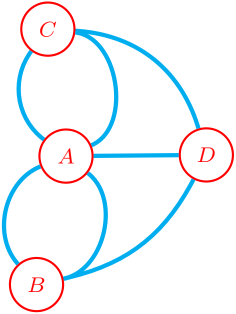
## Computational complexity

Computational resources (time, space, memory) needed to run an algorithm.

## Time complexity

An estimate how the running time scales with the input.

Example: finding the highest degree




| Vertex degrees: | 5 | 3 | 3 | 3 |
|---|---|---|---|---|

Current highest degree: 5

# Complexity

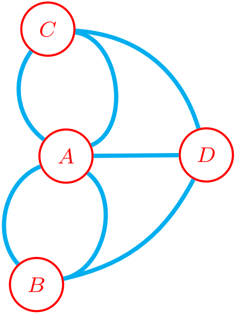
## Computational complexity

Computational resources (time, space, memory) needed to run an algorithm.

## Time complexity

An estimate how the running time scales with the input.

Example: finding the highest degree




| Vertex degrees: | 5 | 3 | 3 | 3 |
|---|---|---|---|---|
| Current highest degree: | | | 5 | |

# Complexity

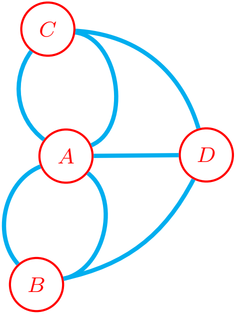
## Computational complexity

Computational resources (time, space, memory) needed to run an algorithm.

## Time complexity

An estimate how the running time scales with the input.

Example: finding the highest degree




| Vertex degrees: | 5 | 3 | 3 | 3 |
|---|---|---|---|---|
| Current highest degree: | | | 5 | |

# Complexity

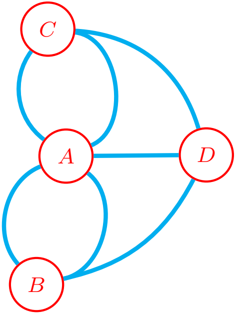
## Computational complexity

Computational resources (time, space, memory) needed to run an algorithm.

## Time complexity

An estimate how the running time scales with the input.

Example: finding the highest degree




| | | | | |
|---|---|---|---|---|
| Vertex degrees: | 5 | 3 | 3 | 3 |
| Highest degree: | | 5 | | |

Time complexity: $\boxed{\mathcal{O}(|V|)}$

# Complexity

## Computational complexity

Computational resources (time, space, memory) needed to run an algorithm.

## Time complexity

An estimate how the running time scales with the input.

Example: $\mathcal{O}(|V|^4)$

| $\|V\|$ | Running time |
|---|---|
| 1000 (test network) | 1 second |

# Complexity

## Computational complexity

Computational resources (time, space, memory) needed to run an algorithm.

## Time complexity

An estimate how the running time scales with the input.

Example: $\mathcal{O}(|V|^4)$

| $\lvert V\rvert$ | Running time |
|---|---|
| 1000 (test network) | 1 second |
| $10^6$ | $\approx 30,000$ years |

# Complexity

## Computational complexity

Computational resources (time, space, memory) needed to run an algorithm.

## Time complexity

An estimate how the running time scales with the input.

Example: $\mathcal{O}(|V|^4)$

| $\lvert V\rvert$ | Running time |
|---|---|
| 1000 (test network) | 1 second |
| $10^6$ | $\approx 30,000$ years |
| $6 \cdot 10^9$ | ??? |

# Complexity

## Computational complexity

Computational resources (time, space, memory) needed to run an algorithm.

## Time complexity

An estimate how the running time scales with the input.

Example: $\mathcal{O}(|V|^4)$

| $\lvert V\rvert$ | Running time |
|---|---|
| 1000 (test network) | 1 second |
| $10^6$ | $\approx 30,000$ years |
| $6 \cdot 10^9$ | ??? |

## Key takeaway

Always pre-estimate the running time (test run first, scale up appropriately).

# Closeness centrality

**Centrality index**

A measure of *importance*, *influence*, or *power* of a vertex/edge in a network.

# Closeness centrality

**Centrality index**

A measure of *importance*, *influence*, or *power* of a vertex/edge in a network.

**Service facility location problem**

Where should we place a shopping mall to minimise the total distance to all customers in the region?

# Closeness centrality

**Centrality index**

A measure of *importance*, *influence*, or *power* of a vertex/edge in a network.

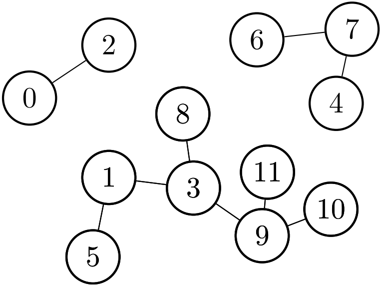
**Service facility location problem**

Where should we place a shopping mall to minimise the total distance to all customers in the region?

**Closeness centrality**

$$\boxed{c_C(u) = \left(\sum_{v \in V} d(u,v)\right)^{-1}}$$

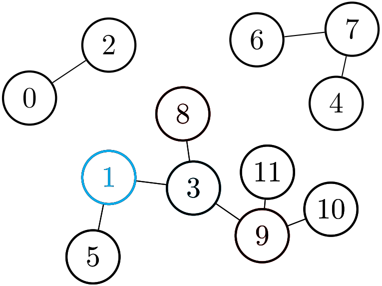
# Breadth-first search




$$\begin{bmatrix}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0
\end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

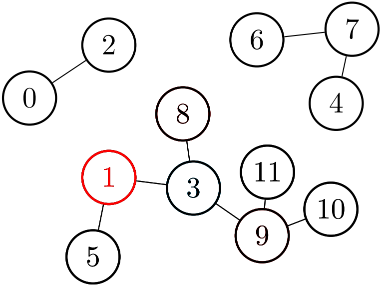
# Breadth-first search




$$\begin{bmatrix}
0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&1&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&1&0&1&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0
\end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

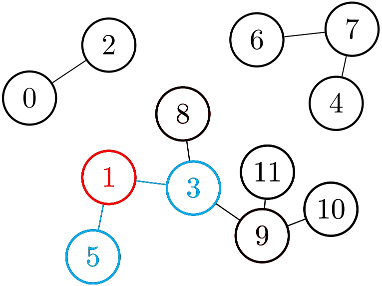
# Breadth-first search




$$\begin{bmatrix}
0&0&1&0&0&0&0&0&0&0&0&0\\
\color{red}0&\color{red}0&\color{red}0&\color{red}1&\color{red}0&\color{red}1&\color{red}0&\color{red}0&\color{red}0&\color{red}0&\color{red}0&\color{red}0\\
1&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&1&0&1&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0
\end{bmatrix}$$

$$\begin{bmatrix}-1 & \color{red}0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1\end{bmatrix}$$

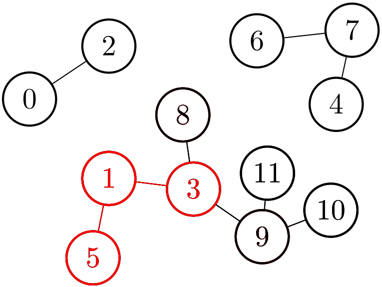
# Breadth-first search




$$\begin{bmatrix} 0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&1&1&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

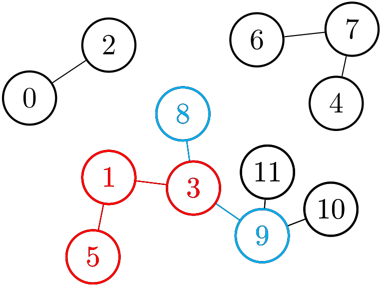
# Breadth-first search




$$\begin{bmatrix} 0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&1&1&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 & -1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

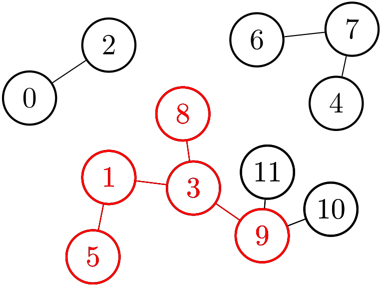
# Breadth-first search




$$\begin{bmatrix} 0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&1&1&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 & -1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

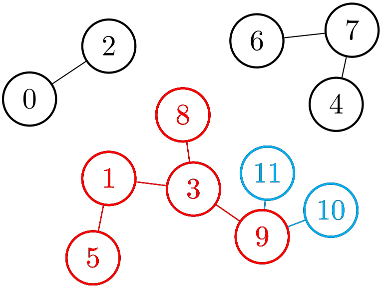
# Breadth-first search




$$\begin{bmatrix} 0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&1&1&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 & -1 & 1 & -1 & 1 & -1 & -1 & 2 & 2 & -1 & -1 \end{bmatrix}$$

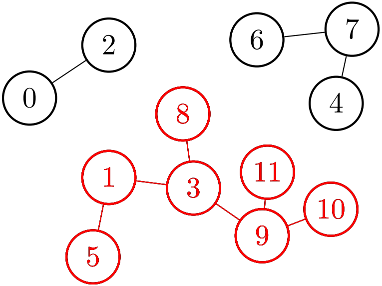
# Breadth-first search




$$\begin{bmatrix} 0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&1&1&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 & -1 & 1 & -1 & 1 & -1 & -1 & 2 & 2 & -1 & -1 \end{bmatrix}$$

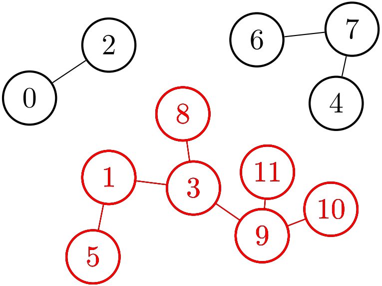
# Breadth-first search




$$\begin{bmatrix} 0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&1&1&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 & -1 & 1 & -1 & 1 & -1 & -1 & 2 & 2 & 3 & 3 \end{bmatrix}$$

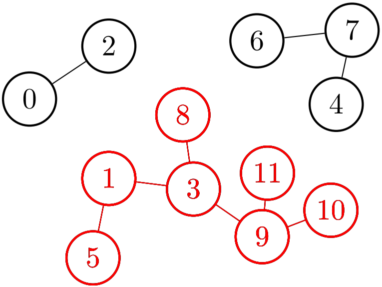
# Breadth-first search




$$\begin{bmatrix} 0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&1&1&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 & -1 & 1 & -1 & 1 & -1 & -1 & 2 & 2 & 3 & 3 \end{bmatrix}$$

# Breadth-first search




$$\begin{bmatrix}
0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&1&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&1&0&1&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0
\end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 & -1 & 1 & -1 & 1 & -1 & -1 & 2 & 2 & 3 & 3 \end{bmatrix}$$

# Breadth-first search

- Recovers connected components

# Breadth-first search

- Recovers connected components
- Can be easily extended to find the corresponding shortest paths as well

# Breadth-first search

- Recovers connected components
- Can be easily extended to find the corresponding shortest paths as well
- Naïve implementation: $\mathcal{O}(|V| + r|V| + |V| \cdot |E|/|V|)$ (worst case: $r = n$; most networks in practice: $r = \log n$)

# Breadth-first search

- Recovers connected components
- Can be easily extended to find the corresponding shortest paths as well
- Naïve implementation: $\mathcal{O}(|V| + r|V| + |V| \cdot |E|/|V|)$ (worst case: $r = n$; most networks in practice: $r = \log n$)
- Optimised code: $\mathcal{O}(|V| + |V| \cdot |E|/|V|)$ (sparse networks: $\mathcal{O}(|V|)$, dense networks: $\mathcal{O}(|V|^2)$)

# Breadth-first search

- Recovers connected components
- Can be easily extended to find the corresponding shortest paths as well
- Naïve implementation: $\mathcal{O}(|V| + r|V| + |V| \cdot |E|/|V|)$ (worst case: $r = n$; most networks in practice: $r = \log n$)
- Optimised code: $\mathcal{O}(|V| + |V| \cdot |E|/|V|)$ (sparse networks: $\mathcal{O}(|V|)$, dense networks: $\mathcal{O}(|V|^2)$)
- Closeness centrality $c_C(u)$ for $u \in V$: $\mathcal{O}(|V| + |E|)$

# Breadth-first search

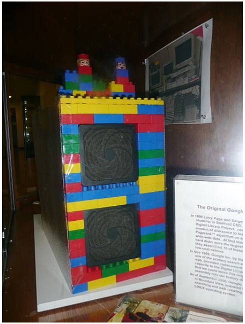
- Recovers connected components
- Can be easily extended to find the corresponding shortest paths as well
- Naïve implementation: $\mathcal{O}(|V| + r|V| + |V| \cdot |E|/|V|)$ (worst case: $r = n$; most networks in practice: $r = \log n$)
- Optimised code: $\mathcal{O}(|V| + |V| \cdot |E|/|V|)$ (sparse networks: $\mathcal{O}(|V|)$, dense networks: $\mathcal{O}(|V|^2)$)
- Closeness centrality $c_C(u)$ for $u \in V$: $\mathcal{O}(|V| + |E|)$
- Shortest paths on weighted networks: Dijkstra's algorithm

# PageRank

## PageRank

$$c_{\mathrm{PR}}(u) = d \sum_{v \in \mathcal{N}^-(u)} \frac{c_{\mathrm{PR}}(v)}{\deg^+(v)} + (1-d)$$

Brin, S. & Page, L. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems* **30**(1–7) (1998), 107–117.



# Motivation for the mPW algorithm

A **proper colouring** is an assignment of colours to the vertices of a graph so that no two adjacent vertices have the same colour, i.e., $c : V \to \{1, 2, \ldots, k\}$ s.t. $c(i) \neq c(j)$ for all $ij \in E$.

# Motivation for the mPW algorithm

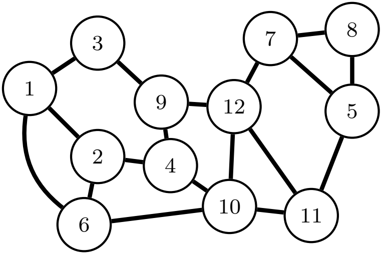
A **proper colouring** is an assignment of colours to the vertices of a graph so that no two adjacent vertices have the same colour, i.e., $c : V \to \{1, 2, \ldots, k\}$ s.t. $c(i) \neq c(j)$ for all $ij \in E$.

A graph that has a $k$-colouring is said to be $k$-**colourable**.

# Motivation for the mPW algorithm

A **proper colouring** is an assignment of colours to the vertices of a graph so that no two adjacent vertices have the same colour, i.e., $c : V \to \{1, 2, \ldots, k\}$ s.t. $c(i) \neq c(j)$ for all $ij \in E$.

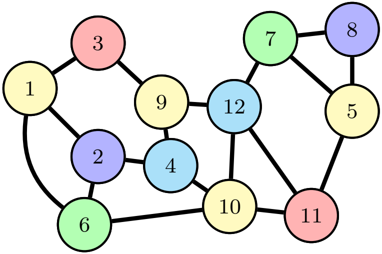
A graph that has a $k$-colouring is said to be $k$-**colourable**.



# Motivation for the mPW algorithm

A **proper colouring** is an assignment of colours to the vertices of a graph so that no two adjacent vertices have the same colour, i.e., $c : V \to \{1, 2, \ldots, k\}$ s.t. $c(i) \neq c(j)$ for all $ij \in E$.

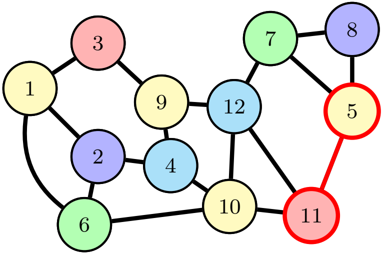
A graph that has a $k$-colouring is said to be $k$-**colourable**.



# Motivation for the mPW algorithm

A **proper colouring** is an assignment of colours to the vertices of a graph so that no two adjacent vertices have the same colour, i.e., $c : V \to \{1, 2, \ldots, k\}$ s.t. $c(i) \neq c(j)$ for all $ij \in E$.

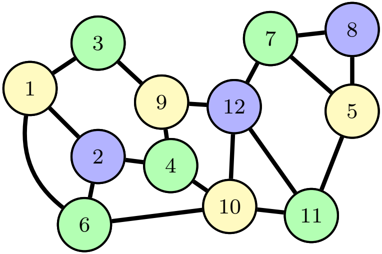
A graph that has a $k$-colouring is said to be $k$-**colourable**.



# Motivation for the mPW algorithm

A **proper colouring** is an assignment of colours to the vertices of a graph so that no two adjacent vertices have the same colour, i.e., $c : V \to \{1, 2, \ldots, k\}$ s.t. $c(i) \neq c(j)$ for all $ij \in E$.

A graph that has a $k$-colouring is said to be $k$-**colourable**.



# Petford–Welsh algorithm

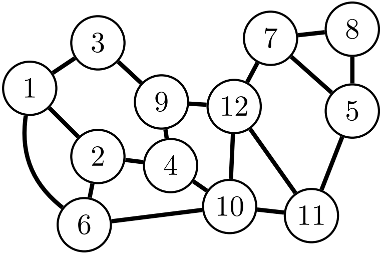
## Petford–Welsh algorithm

Petford, A. D. & Welsh, D. J. A. A Randomised 3-Colouring Algorithm, *Discrete Math.* **74** (1989), 253–261.

Žerovnik, J. A Randomized Algorithm for $k$-Colorability, *Discrete Math.* **131** (1994), 379–393.

# Petford–Welsh algorithm

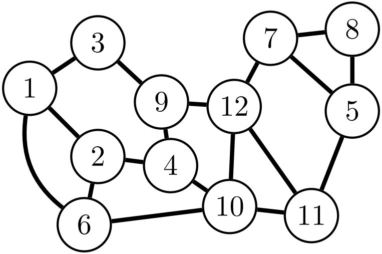
## A randomised $k$-colouring algorithm



# Petford–Welsh algorithm

## A randomised $k$-colouring algorithm

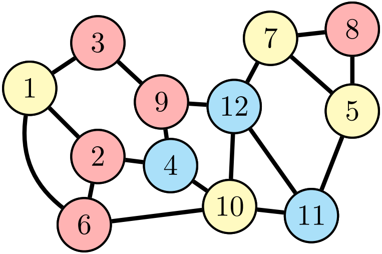
1. get initial $k$-colouring



# Petford–Welsh algorithm

## A randomised $k$-colouring algorithm

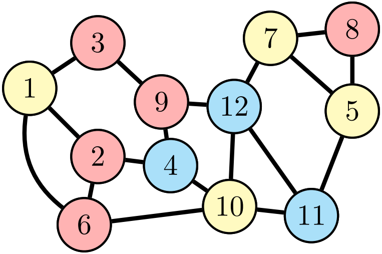
1. get initial $k$-colouring



# Petford–Welsh algorithm

## A randomised $k$-colouring algorithm

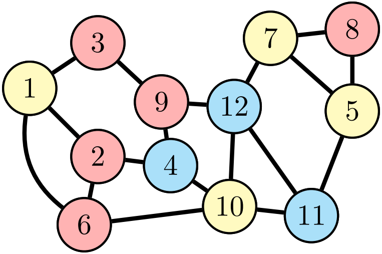
1. get initial $k$-colouring
2. **while** (there is a *bad vertex*) **and** (not too many steps) **repeat**



# Petford–Welsh algorithm

**A randomised $k$-colouring algorithm**

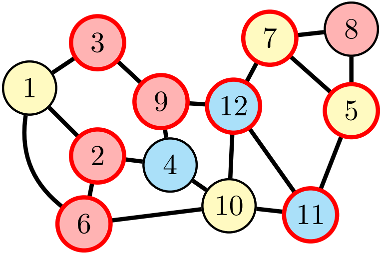
1. get initial $k$-colouring
2. **while** (there is a *bad vertex*) **and** (not too many steps) **repeat**



# Petford–Welsh algorithm

## A randomised $k$-colouring algorithm

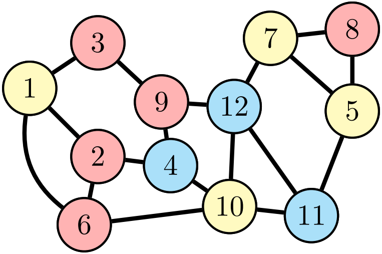
1. get initial $k$-colouring
2. **while** (there is a *bad vertex*) **and** (not too many steps) **repeat**



# Petford–Welsh algorithm

## A randomised $k$-colouring algorithm

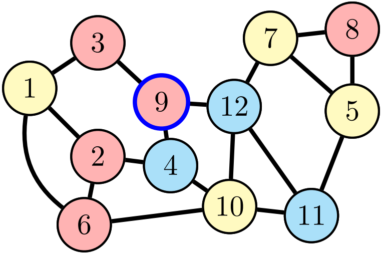
1. get initial $k$-colouring
2. **while** (there is a *bad vertex*) **and** (not too many steps) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random



# Petford–Welsh algorithm

## A randomised $k$-colouring algorithm

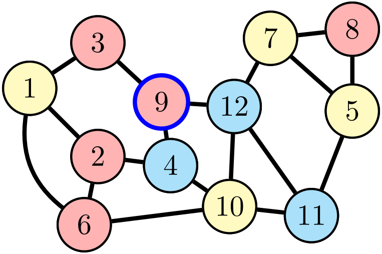
1. get initial $k$-colouring
2. **while** (there is a *bad vertex*) **and** (not too many steps) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random



# Petford–Welsh algorithm

## A randomised $k$-colouring algorithm

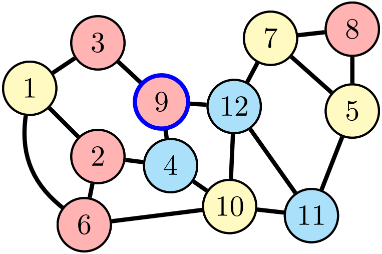
1. get initial $k$-colouring
2. **while** (there is a *bad vertex*) **and** (not too many steps) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{-\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = \#$neighbours of $v$ of colour $i$



# Petford–Welsh algorithm

## A randomised $k$-colouring algorithm

1. get initial $k$-colouring
2. **while** (there is a *bad vertex*) **and** (not too many steps) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{-\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = $ #neighbours of $v$ of colour $i$




$\mathcal{N}(9,r) = 1$

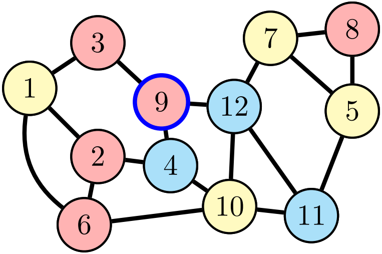
$\mathcal{N}(9,b) = 2$

$\mathcal{N}(9,y) = 0$

# Petford–Welsh algorithm

## A randomised $k$-colouring algorithm

1. get initial $k$-colouring
2. **while** (there is a *bad vertex*) **and** (not too many steps) **repeat**
   - 2.1 choose a bad vertex $v$ uniformly at random
   - 2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{-\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i)$ = #neighbours of $v$ of colour $i$




$\mathcal{N}(9,r) = 1$

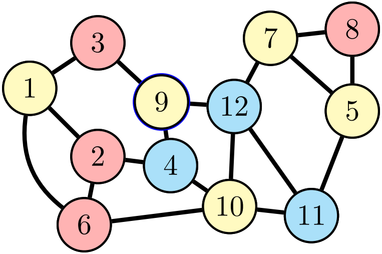
$\mathcal{N}(9,b) = 2$

$\mathcal{N}(9,y) = 0$

# Petford–Welsh algorithm

## A randomised $k$-colouring algorithm

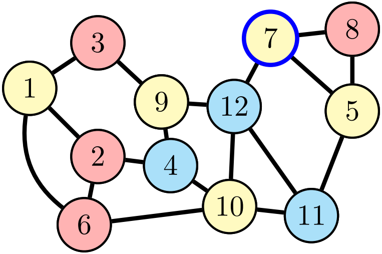
1. get initial $k$-colouring
2. **while** (there is a *bad vertex*) **and** (not too many steps) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{-\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = \#$neighbours of $v$ of colour $i$



# Petford–Welsh algorithm

## A randomised $k$-colouring algorithm

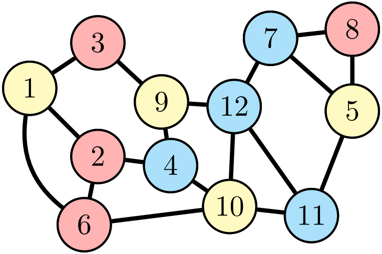
1. get initial $k$-colouring
2. **while** (there is a *bad vertex*) **and** (not too many steps) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{-\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = $ #neighbours of $v$ of colour $i$



# Petford–Welsh algorithm

**A randomised $k$-colouring algorithm**

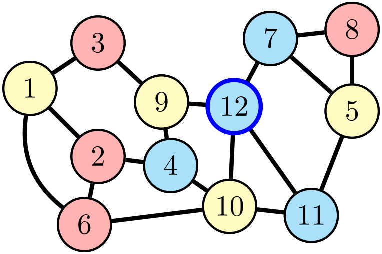
1. get initial $k$-colouring
2. **while** (there is a *bad vertex*) **and** (not too many steps) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{-\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = \#$neighbours of $v$ of colour $i$



# Petford–Welsh algorithm

**A randomised $k$-colouring algorithm**

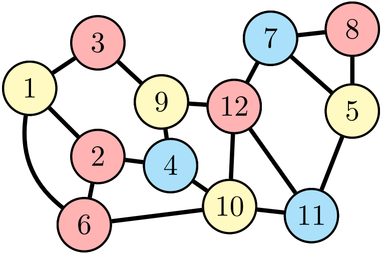
1. get initial $k$-colouring
2. **while** (there is a *bad vertex*) **and** (not too many steps) **repeat**
   - 2.1 choose a bad vertex $v$ uniformly at random
   - 2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{-\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = \#$neighbours of $v$ of colour $i$



# Petford–Welsh algorithm

## A randomised $k$-colouring algorithm

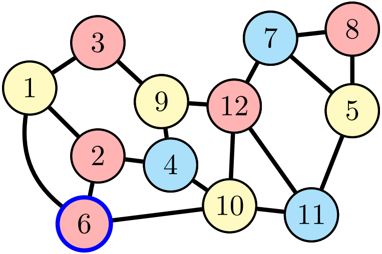
1. get initial $k$-colouring
2. **while** (there is a *bad vertex*) **and** (not too many steps) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{-\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = \#$neighbours of $v$ of colour $i$



# Petford–Welsh algorithm

## A randomised $k$-colouring algorithm

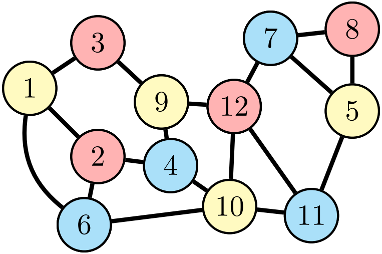
1. get initial $k$-colouring
2. **while** (there is a *bad vertex*) **and** (not too many steps) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{-\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = \#$neighbours of $v$ of colour $i$



# Petford–Welsh algorithm

## A randomised $k$-colouring algorithm

1. get initial $k$-colouring
2. **while** (there is a *bad vertex*) **and** (not too many steps) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{-\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = \#$neighbours of $v$ of colour $i$



# Petford–Welsh algorithm

The Petford–Welsh algorithm ...

- ... mimics the behaviour of a physical process based on a multi-particle system in statistical mechanics (*the antivoter model* by Donnely and Welsh),
- ... acts locally; thus, it is highly parallelisable,
- ... has the *weak convergence property*:
  If $k > \chi(G)$, there is a positive probability that the algorithm finds a proper $k$-colouring in a finite number of steps (regardless of the initial colouring).

# Petford–Welsh Algorithm

**Proposition**

A suitably defined parallel variant of the algorithm with a positive probability finds a proper colouring in one (parallel) step starting from any initial colouring, provided that a proper colouring exists.

**Consequence**

If we increase the number of steps of the algorithm, the probability of reaching a proper colouring becomes as close to 1 as desired.

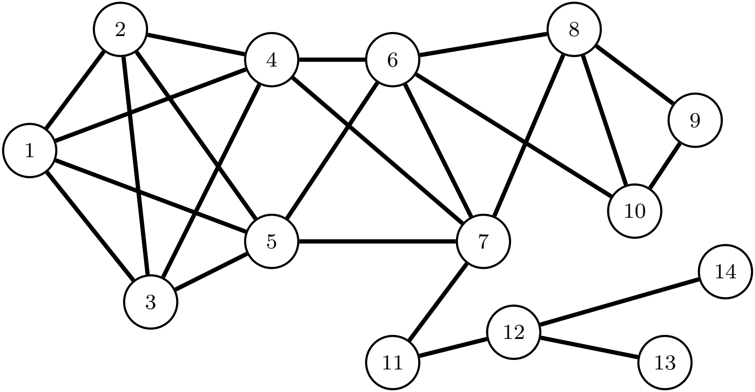
Žerovnik, J. & Kaufman, M. A parallel variant of a heuristical algorithm for graph coloring – Corrigendum, *Parallel Comput.* **18** (1993), 897–900.

# Clustering

Partitioning or grouping data into “similar” subsets.

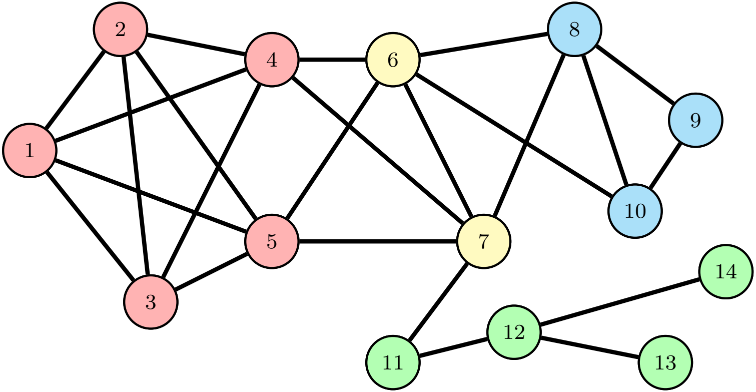
# Clustering

Partitioning or grouping data into “similar” subsets.



# Clustering

Partitioning or grouping data into "similar" subsets.



# Clustering

A *partitioning clustering method* separates a given set of objects $X = \{x_1, x_2, \ldots, x_n\}$ into non-overlapping groups/*clusters* $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ that satisfy

# Clustering

A *partitioning clustering method* separates a given set of objects $X = \{x_1, x_2, \ldots, x_n\}$ into non-overlapping groups/*clusters* $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ that satisfy

- $C_i \neq \emptyset$ for all $1 \leq i \leq m$,

# Clustering

A *partitioning clustering method* separates a given set of objects $X = \{x_1, x_2, \ldots, x_n\}$ into non-overlapping groups/*clusters* $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ that satisfy

- $C_i \neq \emptyset$ for all $1 \leq i \leq m$,
- $\cup_{i=1}^{m} C_i = X$,

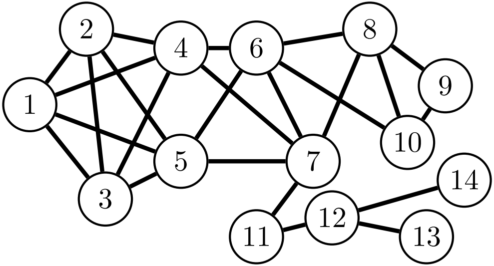
# Clustering

A *partitioning clustering method* separates a given set of objects $X = \{x_1, x_2, \ldots, x_n\}$ into non-overlapping groups/*clusters* $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ that satisfy

- $C_i \neq \emptyset$ for all $1 \leq i \leq m$,
- $\cup_{i=1}^{m} C_i = X$,
- $C_i \cap C_j = \emptyset$ for all $1 \leq i < j \leq m$.

# An adaptation of the Petford–Welsh algorithm

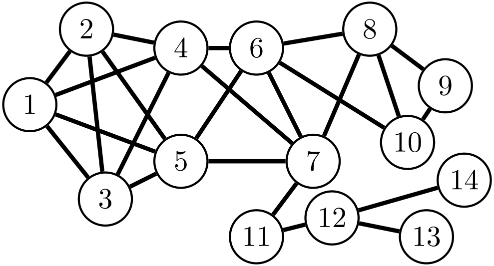
**A randomised *clustering* algorithm [https://github.com/ikicab/mPW]**



# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [`https://github.com/ikicab/mPW`]**

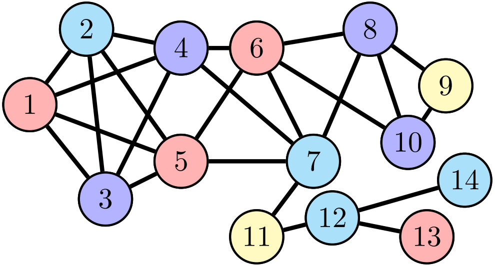
1. get initial $k$-*clustering*



# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [`https://github.com/ikicab/mPW`]**

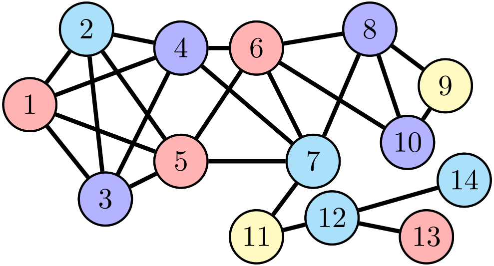
1. get initial $k$-*clustering*



# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [`https://github.com/ikicab/mPW`]**

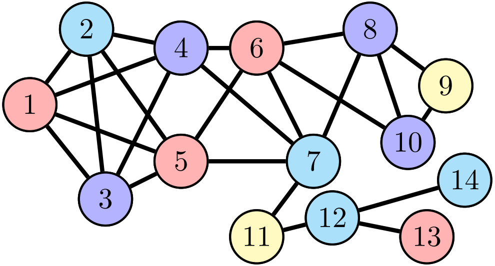
1. get initial $k$-*clustering*
2. **while** (there is a *bad vertex*) **and** $(\mathrm{Var}\left[\textit{bad edges}\right] \geq \texttt{tol})$ **repeat**



# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [`https://github.com/ikicab/mPW`]**

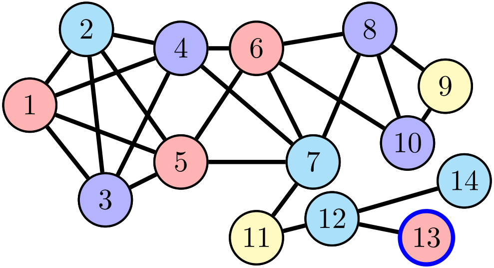
1. get initial $k$-*clustering*
2. **while** (there is a *bad vertex*) **and** $(\mathrm{Var}\,[\textit{bad edges}] \geq \texttt{tol})$ **repeat**
   2.1 choose a bad vertex $v$ uniformly at random



# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [`https://github.com/ikicab/mPW`]**

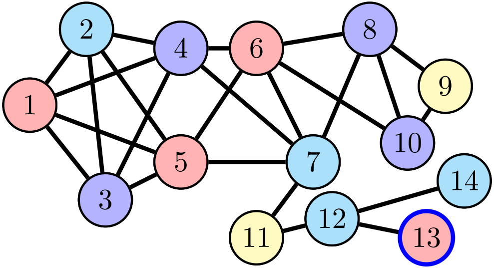
1. get initial $k$-*clustering*
2. **while** (there is a *bad vertex*) **and** ($\mathrm{Var}\left[\textit{bad edges}\right] \geq \texttt{tol}$) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random



# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [`https://github.com/ikicab/mPW`]**

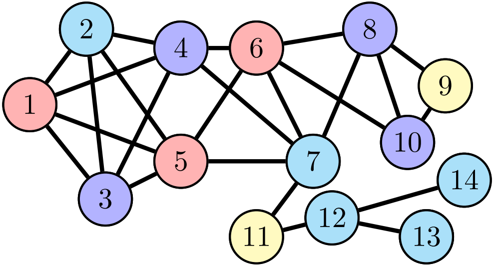
1. get initial $k$-*clustering*
2. **while** (there is a *bad vertex*) **and** ($\mathrm{Var}\left[\textit{bad edges}\right] \geq \texttt{tol}$) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{+\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = $ #neighbours of $v$ of colour $i$



# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [https://github.com/ikicab/mPW]**

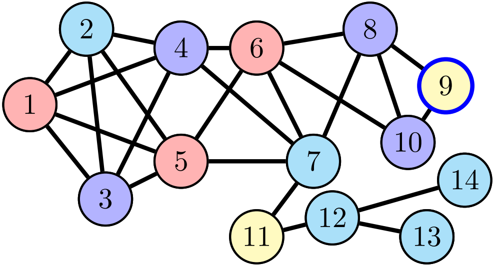
1. get initial $k$-*clustering*
2. **while** (there is a *bad vertex*) **and** ($\mathrm{Var}$ [*bad edges*] $\geq$ `tol`) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{+\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = \#$neighbours of $v$ of colour $i$



# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [`https://github.com/ikicab/mPW`]**

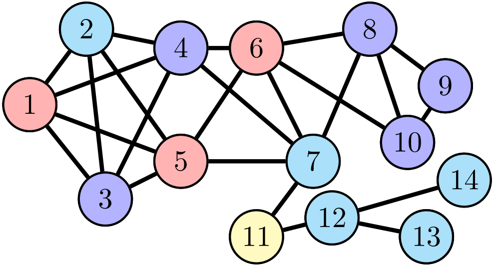
1. get initial $k$-*clustering*
2. **while** (there is a *bad vertex*) **and** ($\mathrm{Var}\left[\textit{bad edges}\right] \geq \texttt{tol}$) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{+\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v, i) = \#$neighbours of $v$ of colour $i$



# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [`https://github.com/ikicab/mPW`]**

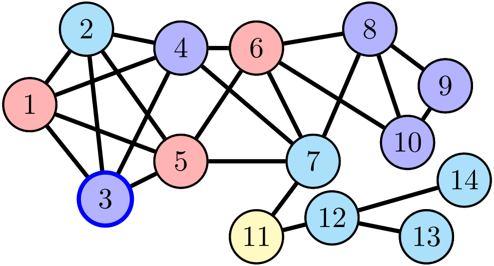
1. get initial $k$-*clustering*
2. **while** (there is a *bad vertex*) **and** ($\mathrm{Var}\left[\textit{bad edges}\right] \geq \texttt{tol}$) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{+\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = \#$neighbours of $v$ of colour $i$



# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [`https://github.com/ikicab/mPW`]**

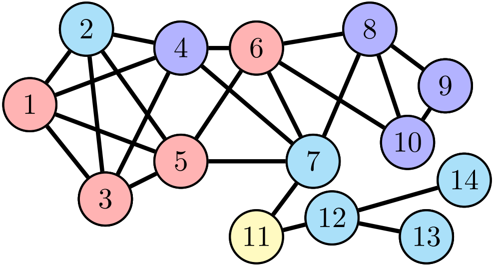
1. get initial $k$-*clustering*
2. **while** (there is a *bad vertex*) **and** ($\mathrm{Var}\left[\textit{bad edges}\right] \geq \texttt{tol}$) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{+\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = \#$neighbours of $v$ of colour $i$



# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [`https://github.com/ikicab/mPW`]**

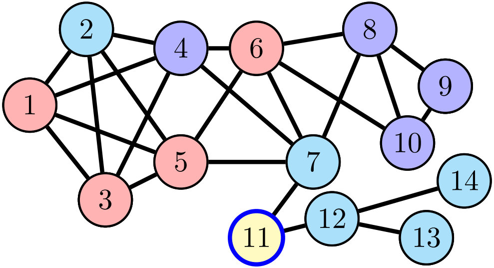
1. get initial $k$-*clustering*
2. **while** (there is a *bad vertex*) **and** $(\mathrm{Var}\left[\textit{bad edges}\right] \geq \mathtt{tol})$ **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{+\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = \#$neighbours of $v$ of colour $i$



# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [`https://github.com/ikicab/mPW`]**

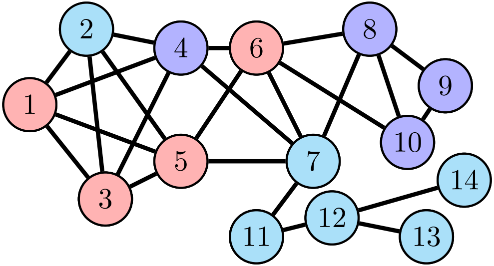
1. get initial $k$-*clustering*
2. **while** (there is a *bad vertex*) **and** ($\mathrm{Var}\left[\textit{bad edges}\right] \geq \texttt{tol}$) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{+\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = $ #neighbours of $v$ of colour $i$



# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [`https://github.com/ikicab/mPW`]**

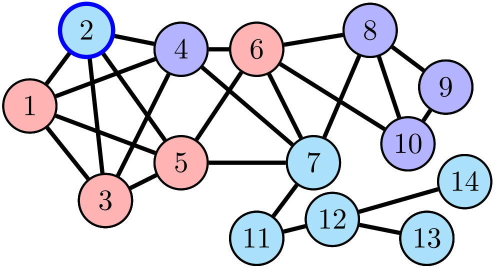
1. get initial $k$-*clustering*
2. **while** (there is a *bad vertex*) **and** ($\mathrm{Var}\,[\textit{bad edges}] \geq \texttt{tol}$) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{+\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = $ #neighbours of $v$ of colour $i$



# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [`https://github.com/ikicab/mPW`]**

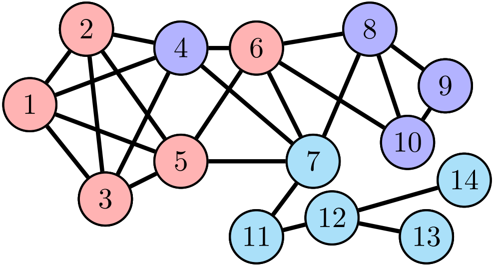
1. get initial $k$-*clustering*
2. **while** (there is a *bad vertex*) **and** ($\mathrm{Var}\left[\textit{bad edges}\right] \geq \texttt{tol}$) **repeat**
   - 2.1 choose a bad vertex $v$ uniformly at random
   - 2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{+\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = $ #neighbours of $v$ of colour $i$



# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [`https://github.com/ikicab/mPW`]**

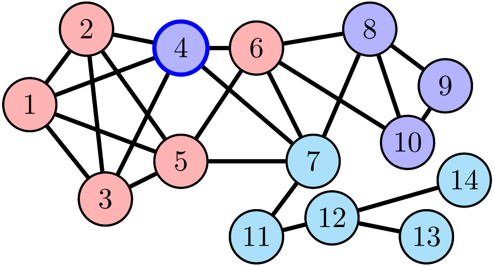
1. get initial $k$-*clustering*
2. **while** (there is a *bad vertex*) **and** ($\mathrm{Var}\,[$*bad edges*$] \geq$ `tol`) **repeat**
   - 2.1 choose a bad vertex $v$ uniformly at random
   - 2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{+\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = \#$neighbours of $v$ of colour $i$



# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [`https://github.com/ikicab/mPW`]**

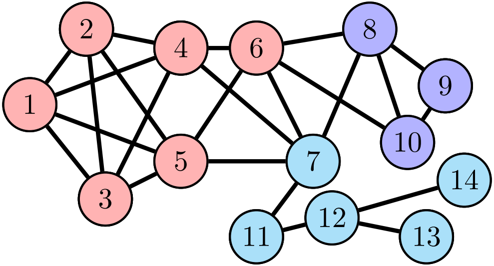
1. get initial $k$-*clustering*
2. **while** (there is a *bad vertex*) **and** ($\mathrm{Var}\,[$*bad edges*$] \geq$ `tol`) **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{+\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = $ #neighbours of $v$ of colour $i$



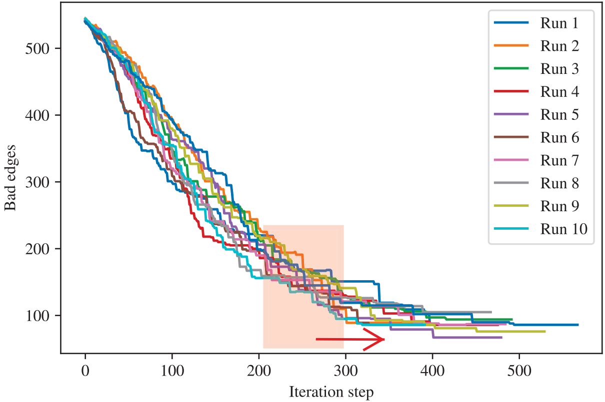
# An adaptation of the Petford–Welsh algorithm

**A randomised *clustering* algorithm [`https://github.com/ikicab/mPW`]**

1. get initial $k$-*clustering*
2. **while** (there is a *bad vertex*) **and** $(\mathrm{Var}\,[\textit{bad edges}] \geq \texttt{tol})$ **repeat**
   2.1 choose a bad vertex $v$ uniformly at random
   2.2 choose a new colour $i$ for $v$ proportionally to $\omega^{+\mathcal{N}(v,i)}$ where $\omega > 1$ and $\mathcal{N}(v,i) = \#$neighbours of $v$ of colour $i$



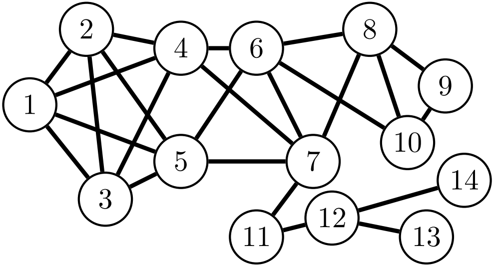
# Stopping condition




$$\mathrm{Var}\left(\texttt{bad\_edges}\left[\texttt{step} - l + 1 : \texttt{step}\right]\right) < \texttt{tol}$$

# Fine-tuning

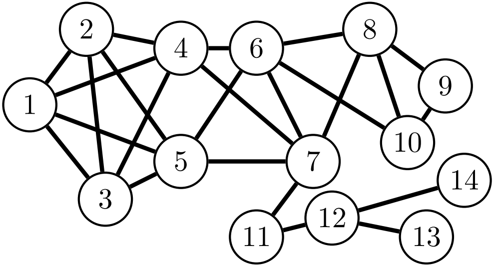
## Problems



# Fine-tuning

## Problems

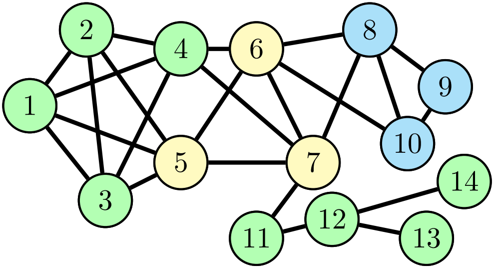
- different clusters get assigned the same colour due to random seeds



# Fine-tuning

### Problems

- different clusters get assigned the same colour due to random seeds

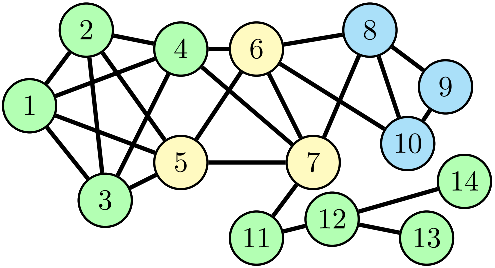


# Fine-tuning

## Problems

- different clusters get assigned the same colour due to random seeds

  [Average *co-membership matrix*: for each clustering solution $c$, $C_c(i,j) = 1$ iff $i$ and $j$ belong to the same cluster (else 0)]

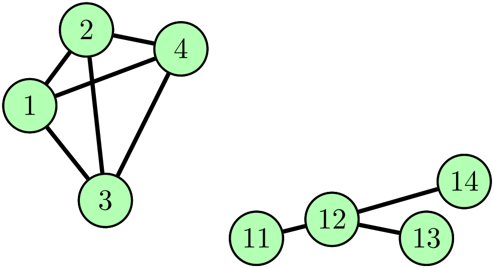


# Fine-tuning

## Problems

- different clusters get assigned the same colour due to random seeds

  [Average *co-membership matrix*: for each clustering solution $c$, $C_c(i,j) = 1$ iff $i$ and $j$ belong to the same cluster (else 0)]

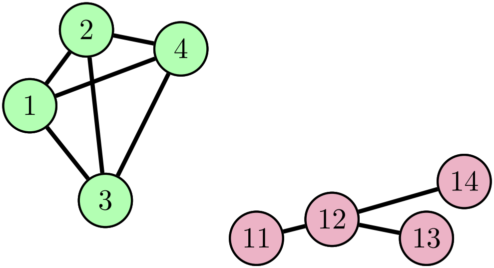


# Fine-tuning

## Problems

- different clusters get assigned the same colour due to random seeds

  [Average *co-membership matrix*: for each clustering solution $c$, $C_c(i, j) = 1$ iff $i$ and $j$ belong to the same cluster (else 0)]

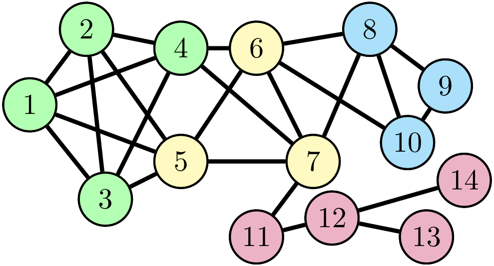


# Fine-tuning

## Problems

- different clusters get assigned the same colour due to random seeds

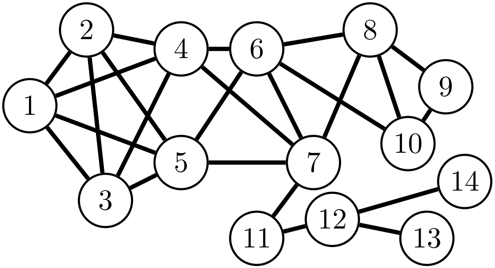
  [Average *co-membership matrix*: for each clustering solution $c$, $C_c(i,j) = 1$ iff $i$ and $j$ belong to the same cluster (else 0)]



# Fine-tuning

## Problems

- different clusters get assigned the same colour due to random seeds
  [Average *co-membership matrix*: for each clustering solution $c$, $C_c(i,j) = 1$ iff $i$ and $j$ belong to the same cluster (else 0)]
- outliers (singleton clusters)

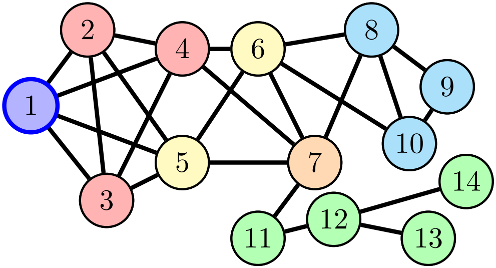


# Fine-tuning

## Problems

- different clusters get assigned the same colour due to random seeds

  [Average *co-membership matrix*: for each clustering solution $c$, $C_c(i,j) = 1$ iff $i$ and $j$ belong to the same cluster (else 0)]
- outliers (singleton clusters)

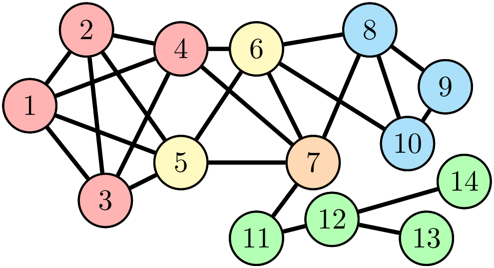


# Fine-tuning

## Problems

- different clusters get assigned the same colour due to random seeds

  [Average *co-membership matrix*: for each clustering solution $c$, $C_c(i,j) = 1$ iff $i$ and $j$ belong to the same cluster (else 0)]
- outliers (singleton clusters)

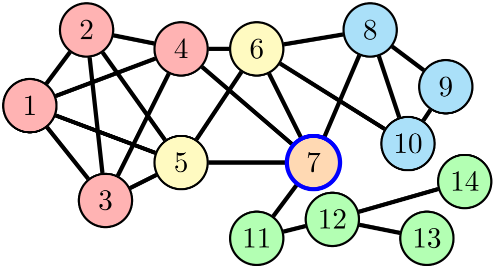


# Fine-tuning

## Problems

- different clusters get assigned the same colour due to random seeds

  [Average *co-membership matrix*: for each clustering solution $c$, $C_c(i,j) = 1$ iff $i$ and $j$ belong to the same cluster (else 0)]
- outliers (singleton clusters)

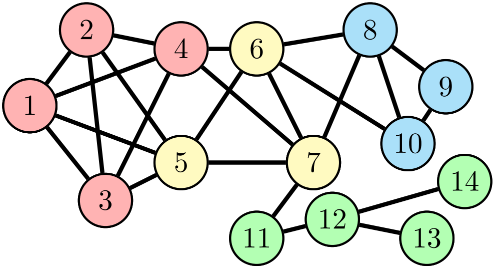


# Fine-tuning

## Problems

- different clusters get assigned the same colour due to random seeds

  [Average *co-membership matrix*: for each clustering solution $c$, $C_c(i,j) = 1$ iff $i$ and $j$ belong to the same cluster (else 0)]
- outliers (singleton clusters)



# Quality measures

## Internal indices

The clustering is judged on the basis of certain intrinsic statistical properties of the clustering itself.

*Modularity, conductance, coverage*

# Quality measures

**Internal indices**

The clustering is judged on the basis of certain intrinsic statistical properties of the clustering itself.

*Modularity, conductance, coverage*

**External indices**

The clustering is compared to a user-given gold-standard clustering (using a pairwise/mapping approach).

*Normalised mutual information, adjusted mutual information, adjusted Rand index,* $F_\beta$ *score, Fowlkes–Mallows index, Jaccard index, V-measure*

# Quality measures / Internal indices

## Modularity

$$\boxed{Q = \frac{1}{2|E|} \sum_{u,v \in V} \left( a_{uv} - \frac{k_u k_v}{2|E|} \right) \delta(c_u, c_v)}$$

Compares the presence of each intra-cluster edge with the probability of this edge in a random graph

# Quality measures / Internal indices

## Modularity

$$\boxed{Q = \frac{1}{2|E|} \sum_{u,v \in V} \left( a_{uv} - \frac{k_u k_v}{2|E|} \right) \delta(c_u, c_v)}$$

Compares the presence of each intra-cluster edge with the probability of this edge in a random graph

## Coverage

$$\boxed{\gamma = \frac{\sum_{u,v \in V} a_{uv} \delta(c_u, c_v)}{\sum_{u,v \in V} a_{uv}}}$$

A measure of intra-cluster density

# Quality measures / Internal indices

## Conductance

$$\boxed{\phi = 1 - \frac{1}{|\mathcal{C}|} \sum_{C_i \in \mathcal{C}} \phi(C_i)}$$

$$\phi(C_i) = \frac{\sum_{u \in C_i, v \notin C_i} a_{uv}}{\min\left\{\sum_{u \in C_i, v \in V} a_{uv}, \sum_{u \notin C_i, v \in V} a_{uv}\right\}}$$

A measure of inter-cluster sparsity

# Quality measures / External indices

## Normalized mutual information

$$\boxed{\mathrm{NMI}(\mathcal{C},\mathcal{G}) = \frac{\mathrm{MI}(\mathcal{C},\mathcal{G})}{\sqrt{\mathrm{H}(\mathcal{C})\mathrm{H}(\mathcal{G})}}}$$

$$\begin{aligned}
\mathrm{MI}(\mathcal{C},\mathcal{G}) &= \mathrm{H}(\mathcal{C}) + \mathrm{H}(\mathcal{G}) - \mathrm{H}(\mathcal{C},\mathcal{G}) \\
\mathrm{H}(\mathcal{C}_i) &= -\sum_{C\in\mathcal{C}_i} \frac{|C|}{|V|}\log\frac{|C|}{|V|} \\
\mathrm{H}(\mathcal{C},\mathcal{G}) &= -\sum_{C_i\in\mathcal{C},G_j\in\mathcal{G}} \frac{|C_i\cap G_j|}{|V|}\log\frac{|C_i\cap G_j|}{|V|}
\end{aligned}$$

A measure of “information overlap” between $\mathcal{C}$ and $\mathcal{G}$

# Quality measures / External indices

## Adjusted mutual information

$$\boxed{\mathrm{AMI} = \frac{\mathrm{MI}(\mathcal{C},\mathcal{G}) - \mathbb{E}[\mathrm{MI}(\mathcal{C},\mathcal{G})]}{\sqrt{\mathrm{H}(\mathcal{C})\mathrm{H}(\mathcal{G})} - \mathbb{E}[\mathrm{MI}(\mathcal{C},\mathcal{G})]}}$$

A measure of "information overlap" between $\mathcal{C}$ and $\mathcal{G}$ adjusted for chance

# Quality measures / External indices

## Adjusted Rand index

$$\boxed{\begin{aligned}\mathrm{ARI}(\mathcal{C},\mathcal{G}) &= \frac{\mathrm{RI}(\mathcal{C},\mathcal{G}) - \mathrm{E}[\mathrm{RI}(\mathcal{C},\mathcal{G})]}{\max(\mathrm{RI}(\mathcal{C},\mathcal{G})) - \mathrm{E}[\mathrm{RI}(\mathcal{C},\mathcal{G})]} = \\ &= \frac{2(TP \cdot TN - FP \cdot FN)}{(TN+FP)(FP+TP)+(TN+FN)(FN+TP)}\end{aligned}}$$

A measure of the level of agreement between $\mathcal{C}$ and $\mathcal{G}$ as the fraction of agreeing pairs of vertices to all possible pairs of vertices

# Quality measures / External indices

**$F_\beta$ score**

$$\boxed{F_\beta = \frac{(1+\beta^2)\cdot TP}{(1+\beta^2)\cdot TP + \beta^2\cdot FN + FP}}$$

Weighted harmonic mean of precision and recall

# Quality measures / External indices

## $F_\beta$ score

$$\boxed{F_\beta = \frac{(1+\beta^2)\cdot TP}{(1+\beta^2)\cdot TP + \beta^2\cdot FN + FP}}$$

Weighted harmonic mean of precision and recall

## Fowlkes–Mallows index

$$\boxed{\mathrm{FM} = \sqrt{\frac{TP}{TP+FP}\cdot\frac{TP}{TP+FN}}}$$

Geometric mean of precision and recall

# Quality measures / External indices

## Jaccard index

$$\boxed{F_\beta = \frac{(1+\beta^2)\cdot TP}{(1+\beta^2)\cdot TP + \beta^2 \cdot FN + FP}}$$

# Quality measures / External indices

## Jaccard index

$$\boxed{F_\beta = \frac{(1+\beta^2)\cdot TP}{(1+\beta^2)\cdot TP + \beta^2 \cdot FN + FP}}$$

## V-measure

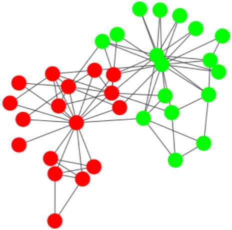
$$\boxed{\mathrm{V}_\beta = (1+\beta)\frac{ho \cdot cp}{\beta \cdot ho + cp}}$$

Harmonic mean of homogeneity $ho$ and completeness $cp$ of the clustering solution

# Experiments

**Zachary ($|V| = 34, |E| = 78$)**

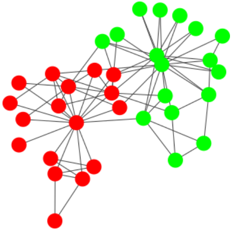
Ties amongst the members of a university karate club by Wayne Zachary.

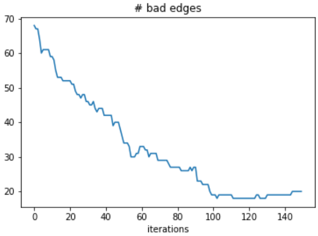


# Experiments

**Zachary ($|V| = 34, |E| = 78$)**

Ties amongst the members of a university karate club by Wayne Zachary.





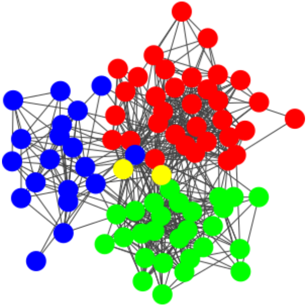
# Experiments / Zachary's karate club

| Method | NMI | ARI | $\phi$ | $\gamma$ | $Q$ | $\lvert\mathcal{C}\rvert$ |
|---|---|---|---|---|---|---|
| Edge bet. | 0.517 | 0.392 | 0.424 | 0.692 | 0.401 | 5 |
| Fastgreedy | 0.576 | 0.568 | 0.574 | 0.756 | 0.381 | 3 |
| Infomap | 0.578 | 0.591 | 0.668 | 0.821 | 0.402 | 3 |
| Label prop. | 0.865 | 0.882 | **0.773** | **0.949** | 0.415 | 3 |
| Leading eig. | 0.612 | 0.435 | 0.487 | 0.667 | 0.393 | 4 |
| Multilevel | 0.516 | 0.392 | 0.558 | 0.731 | 0.419 | 4 |
| Spinglass | 0.627 | 0.509 | 0.563 | 0.756 | **0.420** | 4 |
| Walktrap | 0.531 | 0.321 | 0.434 | 0.590 | 0.353 | 5 |
| **mPW** | **1.000** | **1.000** | **0.773** | **0.949** | 0.403 | 2 |

# Experiments

**UK faculty ($|V| = 34, |E| = 78$)**

The personal friendship network of a faculty of a UK university; the school affiliation of each individual is stored as a vertex attribute.



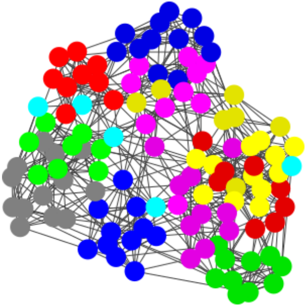
# Experiments / UK faculty

| Method | NMI | ARI | $\phi$ | $\gamma$ | $Q$ | $\lvert\mathcal{C}\rvert$ |
|---|---|---|---|---|---|---|
| Edge bet. | 0.796 | 0.825 | 0.513 | 0.827 | 0.413 | 4 |
| Fastgreedy | 0.849 | 0.820 | 0.553 | 0.775 | 0.444 | 4 |
| Infomap | 0.862 | 0.875 | 0.709 | 0.841 | 0.432 | 3 |
| Label prop. | 0.862 | 0.875 | 0.709 | **0.953** | 0.432 | 3 |
| Leading eig. | 0.863 | 0.871 | 0.488 | 0.768 | 0.397 | 4 |
| Multilevel | 0.802 | 0.796 | 0.573 | 0.749 | **0.449** | 4 |
| Spinglass | 0.872 | 0.842 | 0.573 | 0.749 | **0.449** | 4 |
| Walktrap | 0.862 | 0.875 | 0.709 | 0.841 | 0.432 | 3 |
| **mPW** | **0.911** | **0.918** | **0.741** | **0.953** | 0.432 | 3 |

# Experiments

**American college football ($|V| = 115, |E| = 613$)**

A network of regular season games between teams divided into 12 conferences.



# Experiments / American college football

| Method | NMI | ARI | $\phi$ | $\gamma$ | $Q$ | $\lvert\mathcal{C}\rvert$ |
|---|---|---|---|---|---|---|
| Edge bet. | 0.880 | 0.778 | 0.533 | 0.710 | 0.600 | 10 |
| Fastgreedy | 0.708 | 0.474 | 0.567 | 0.731 | 0.550 | 6 |
| Multilevel | 0.891 | 0.807 | 0.547 | 0.708 | **0.605** | 10 |
| Leading eig. | 0.703 | 0.464 | 0.456 | 0.641 | 0.493 | 8 |
| Infomap | 0.924 | 0.897 | 0.505 | 0.690 | 0.601 | 12 |
| Label prop. | 0.927 | 0.889 | 0.568 | 0.741 | **0.605** | 11 |
| Spinglass | 0.929 | **0.900** | 0.563 | 0.728 | **0.605** | 11 |
| Walktrap | 0.888 | 0.815 | 0.547 | 0.705 | 0.603 | 10 |
| **mPW** | **0.936** | **0.900** | **0.600** | **0.780** | 0.603 | 9 |

# Experiments

**Political blogs ($|V| = 1222, |E| = 16714$)**

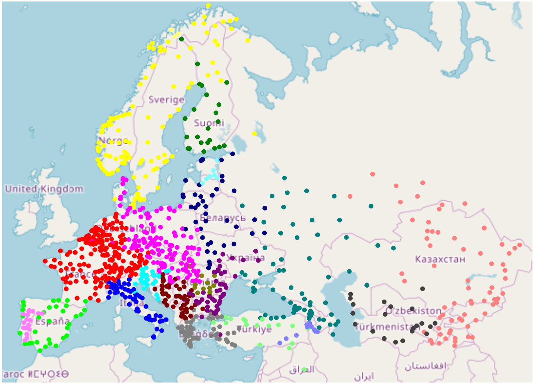
Interactions between liberal and conservative blogs over the period of two months preceding the U.S. Presidential Election of 2004.

| Method | NMI | ARI | $\phi$ | $\gamma$ | $Q$ | $\lvert\mathcal{C}\rvert$ |
|---|---|---|---|---|---|---|
| Edge bet. | – | – | – | – | – | – |
| Fastgreedy | 0.659 | 0.785 | 0.451 | 0.923 | **0.427** | 10 |
| Infomap | 0.523 | 0.651 | 0.250 | 0.899 | 0.423 | 41 |
| Label prop. | 0.723 | 0.813 | **0.857** | **1.000** | 0.426 | 3 |
| Leading eig. | 0.693 | 0.781 | 0.854 | 0.926 | 0.424 | 2 |
| Multilevel | 0.651 | 0.774 | 0.476 | 0.920 | **0.427** | 9 |
| Spinglass | 0.649 | 0.783 | 0.315 | 0.922 | **0.427** | 15 |
| Walktrap | 0.646 | 0.760 | 0.484 | 0.925 | 0.425 | 11 |
| **mPW** | **0.732** | **0.820** | **0.857** | 0.927 | 0.426 | 4 |

# Experiments

**International E-road network ($|V| = 1040, |E| = 1305$)**

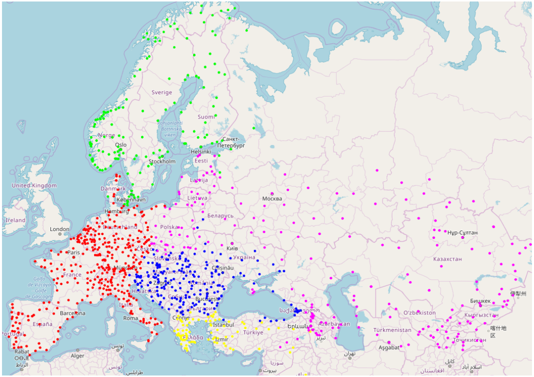
An international system for numbering and designating roads stretching throughout Europe and some parts of Central Asia.



# Experiments

## International E-road network ($|V| = 1040, |E| = 1305$)

An international system for numbering and designating roads stretching throughout Europe and some parts of Central Asia.



# Experiments / International E-road network

| Method | $\phi$ | $\gamma$ | $Q$ | $\lvert\mathcal{C}\rvert$ |
|---|---|---|---|---|
| Edge bet. | – | – | – | – |
| Fastgreedy | 0.860 | 0.917 | 0.861 | 24 |
| Infomap | 0.663 | 0.787 | 0.777 | 126 |
| Label prop. | 0.731 | 0.856 | 0.828 | 82 |
| Leading eig. | 0.794 | 0.887 | 0.835 | 26 |
| Multilevel | 0.873 | 0.921 | 0.867 | 24 |
| Spinglass | 0.866 | 0.924 | **0.872** | 25 |
| Walktrap | 0.757 | 0.886 | 0.828 | 67 |
| **mPW** | **0.945** | **0.979** | 0.845 | 17 |

# Experiments

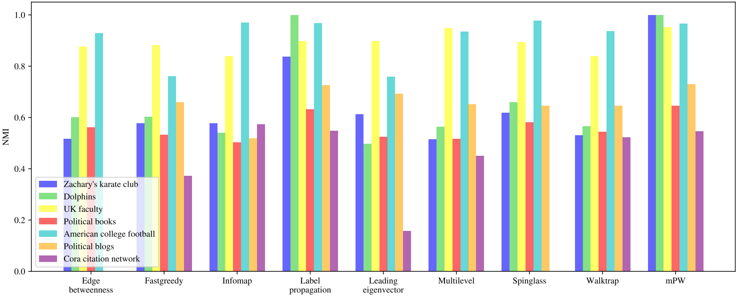
**U.S. airports ($|V| = 745, |E| = 4618$)**

A network of flights between U.S. airports.

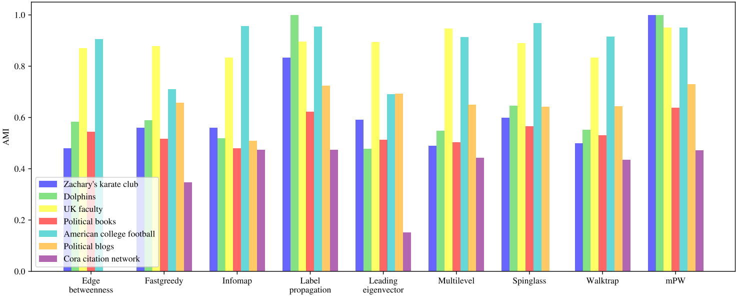
# Experiments / U.S. airports

| Method | $\phi$ | $\gamma$ | $Q$ | $\lvert\mathcal{C}\rvert$ |
|---|---|---|---|---|
| Edge bet. | 0.155 | 0.932 | 0.314 | 118 |
| Fastgreedy | 0.594 | 0.771 | 0.431 | 18 |
| Infomap | 0.477 | 0.913 | 0.310 | 49 |
| Label prop. | 0.653 | 0.959 | 0.258 | 20 |
| Leading eig. | 0.682 | 0.806 | 0.410 | 3 |
| Multilevel | 0.617 | 0.790 | **0.441** | 16 |
| Spinglass | 0.586 | 0.773 | **0.441** | 17 |
| Walktrap | 0.342 | 0.788 | 0.337 | 84 |
| **mPW** | **0.774** | **0.976** | 0.285 | 13 |

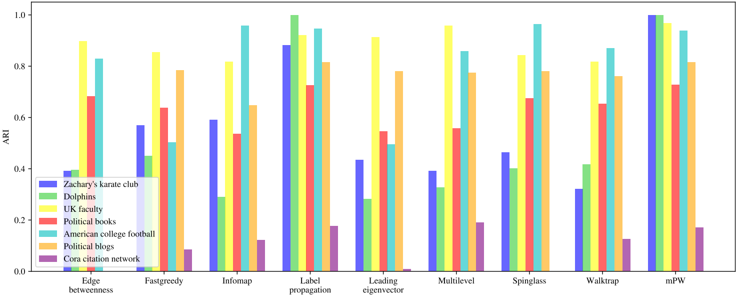
# Experiments / Normalised mutual information



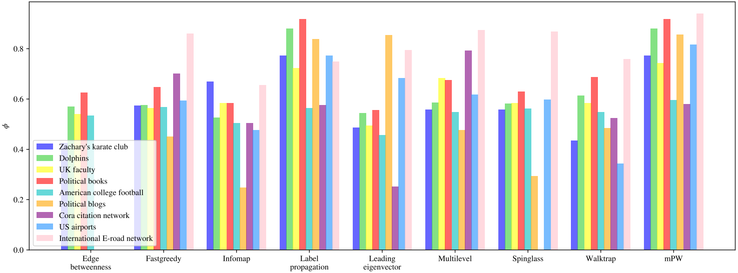
# Experiments / Adjusted mutual information



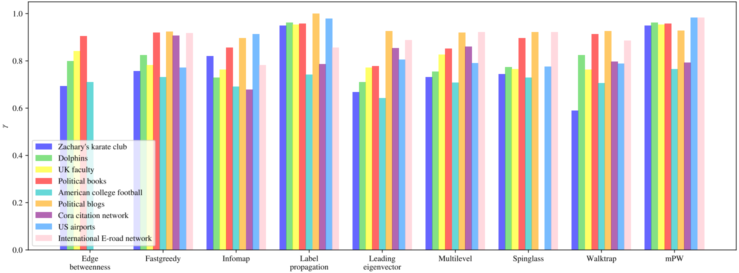
# Experiments / Adjusted Rand index



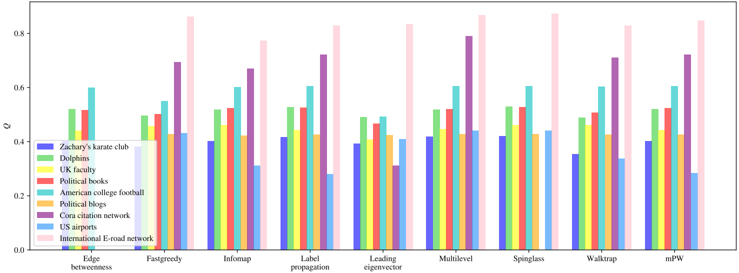
# Experiments / Conductance



# Experiments / Coverage

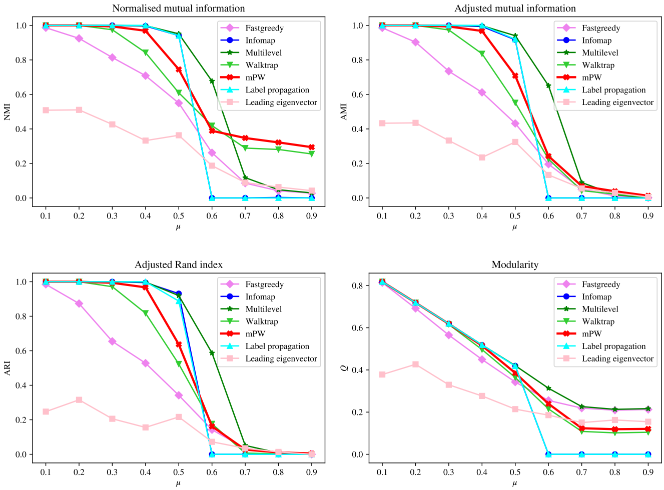


# Experiments / Modularity



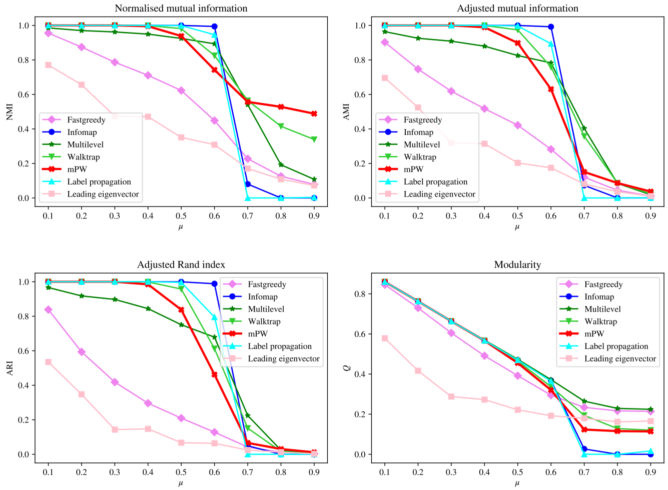
# LFR benchmark

$\mathrm{LFR}(|V| = 1000, \gamma = 2, \beta = 1, \texttt{k\_avg} = 15, \texttt{k\_max} = 100, \texttt{c\_min} = 50, \texttt{c\_max} = 100)$



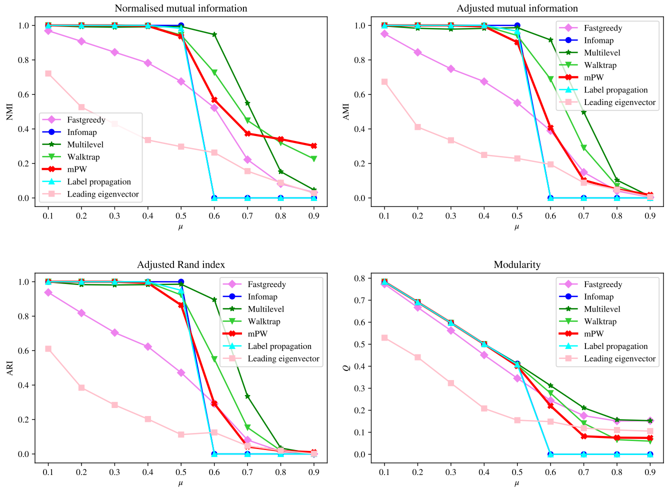
# LFR benchmark

$\mathrm{LFR}(|V| = 1000, \gamma = 3, \beta = 2, \texttt{k\_avg} = 15, \texttt{k\_max} = 50)$



# LFR benchmark

$\mathrm{LFR}(|V| = 1000, \gamma = 2, \beta = 1, \texttt{k\_avg} = 25, \texttt{k\_max} = 150)$



# Computer resources

## Network analysis and visualisation software

- **Pajek** (free; large network analysis): `http://vlado.fmf.uni-lj.si/pub/networks/pajek/`
- **Gephi** (free; (dynamic) network visualisation): `https://gephi.org/`
- **igraph** (free; R/Python/Mathematica/C/C++ network analysis package): `https://igraph.org/`
- **NetworkX** (free; Python package for complex networks): `https://networkx.github.io/`
- **SNAP** (free; Python/C++ high performance library for large networks): `http://snap.stanford.edu/`
- **Mathematica** (commercial): `https://reference.wolfram.com/language/guide/GraphsAndNetworks.html`
- **MATLAB** (commercial): `https://mathworks.com/help/matlab/graph-and-network-algorithms.html`

## Network datasets

- **Newman**: `http://www-personal.umich.edu/~mejn/netdata/`
- **Koblenz Network Collection**: `http://konect.uni-koblenz.de/networks/`
- **SuiteSparse Matrix Collection**: `https://sparse.tamu.edu/`
- **Network Repository**: `http://networkrepository.com/`
- **(BIO)SNAP**: `http://snap.stanford.edu/data/index.html`

# Reading


- Newman, M. E. J. *Networks: An Introduction* (Oxford University Press, New York, NY, 2010).
- Brandes, U. & Erlebach, T. *Network Analysis: Methodological Foundations* (Springer, Berlin, Heidelberg, 2005).
- Ikica, B. *Clustering via the Modified Petford–Welsh Algorithm*. To appear in Ars Mathematica Contemporanea (AMC).
- Ikica, B., Povh, J. & Žerovnik, J. *Clustering as a Dual Problem to Colouring*. Submitted.