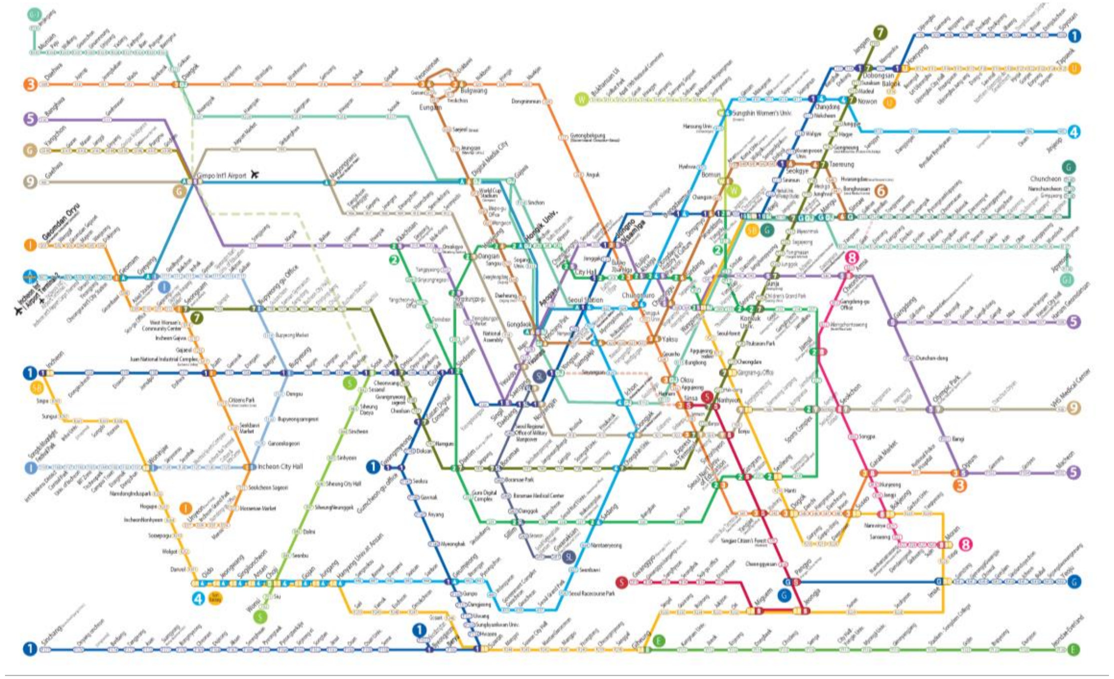
On Improving the Cohesiveness of Graphs by Merging Nodes: Formulation, Analysis, and Algorithms

Fanchen Bu

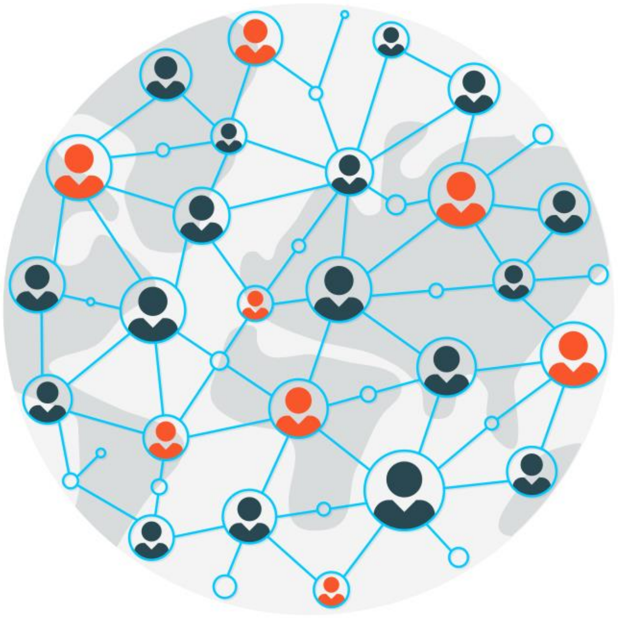
Kijung Shin

Graphs

- A **graph** $G = (V, E)$ consists of a node set V and an edge set E
  - Each edge joins a pair of nodes
- Graphs naturally represent **relations between real-world objects**



Public Transportation Networks



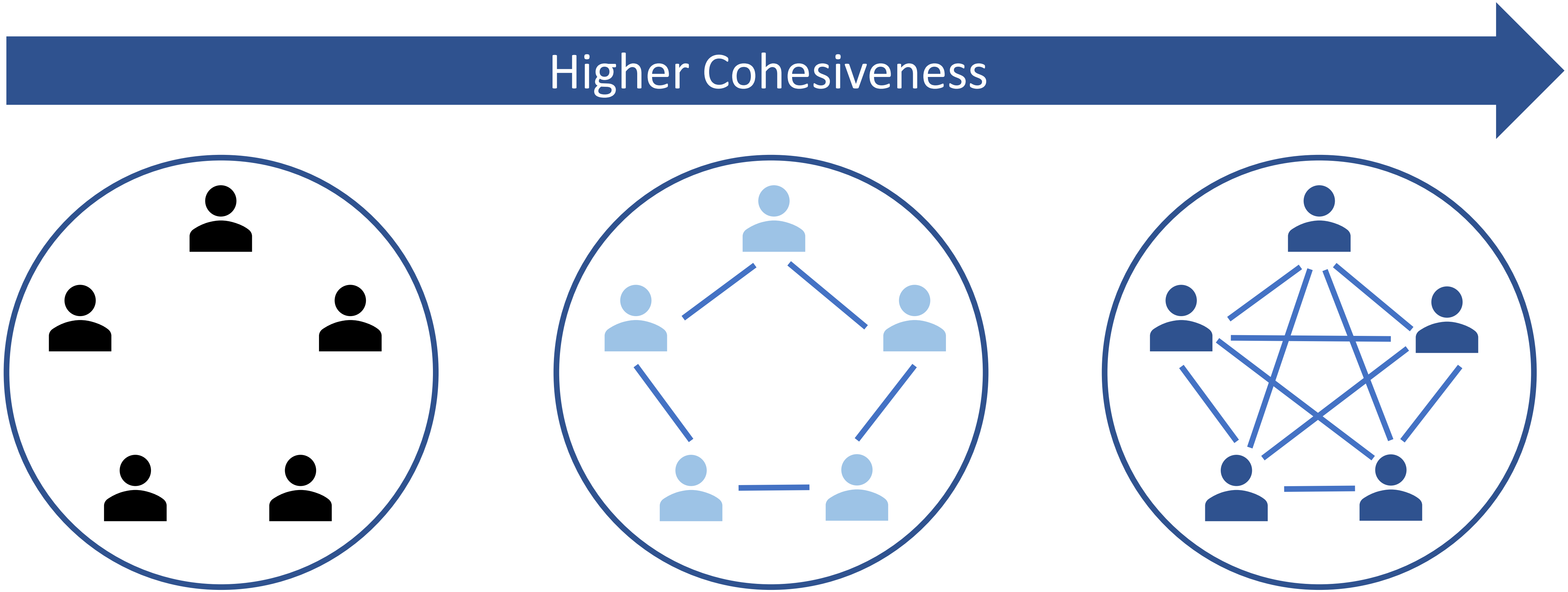
Social Networks

Cohesiveness of Graphs

- **Cohesive in general**: “united and working together effectively”

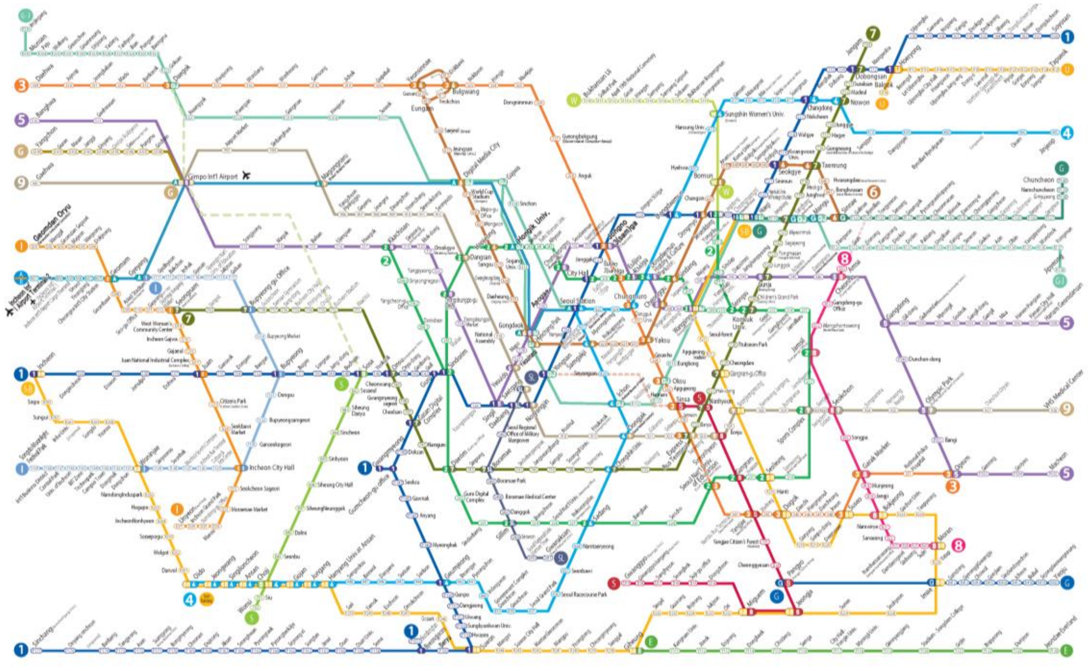
Cohesiveness of Graphs

- **Cohesive graphs**:
  - Intuitively, well-connected graphs have higher cohesiveness

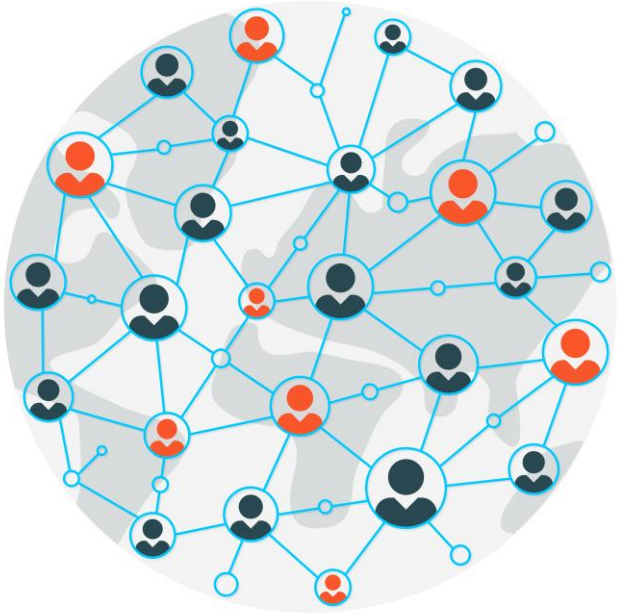


We Love Cohesive Graphs!

- **Cohesiveness = Well-Connectedness + Robustness** 😍



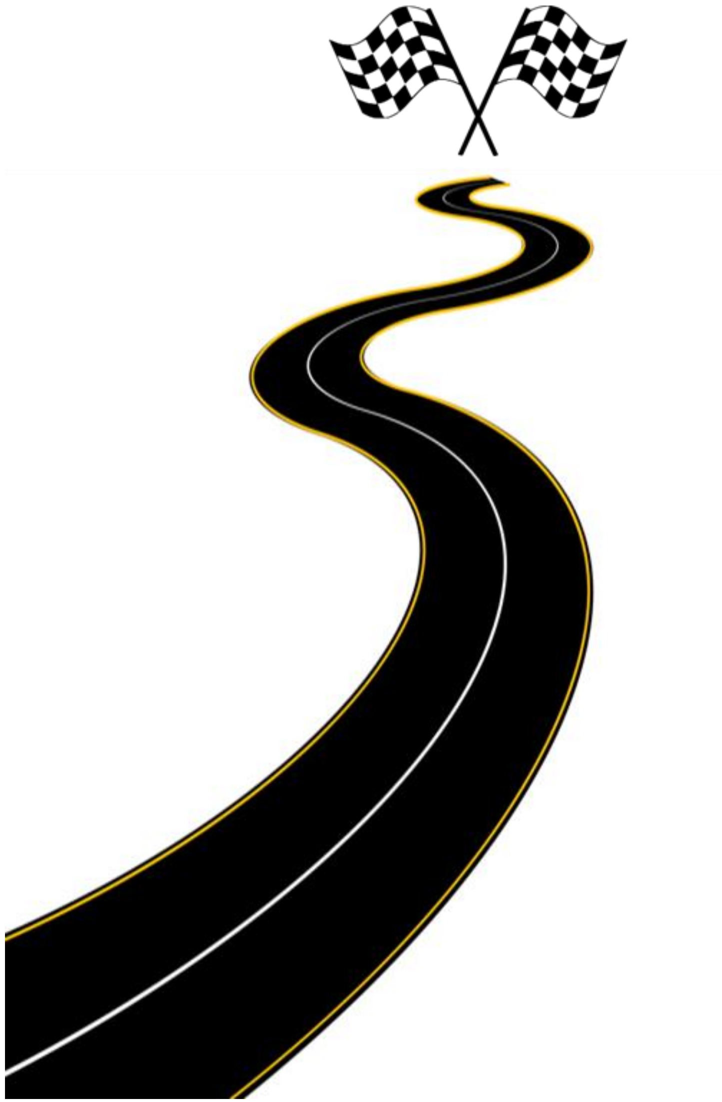
Public Transportation Networks



Social Networks

Roadmap

- **Formulation <<**
- Analysis & Algorithms
- Experiments
- Conclusions



Existing Research

- How can we **improve the cohesiveness** of a network?
  - **Metric**: What to optimize?
  - **Operation**: How to optimize?

		Operation: Anchoring nodes	Operation: Adding edges
Metric	Maximizing the **size of a k-core**	Bhawalkar et al. 2015 Zhang et al. 2022	Zhou et al. 2022
	Maximizing the **size of a k-truss**	Zhang et al. 2018	Sun et al. 2021 Chen et al. 2022

Merging Nodes: Together! Stronger!

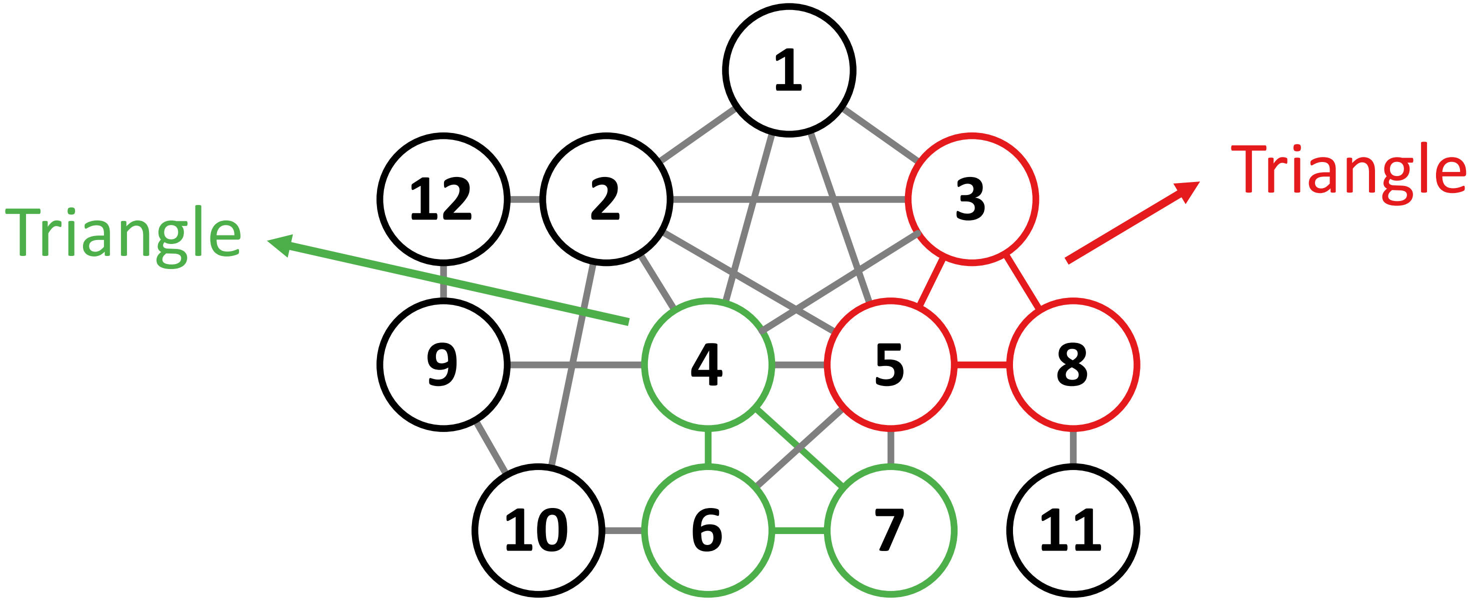
- Merging Stations = More Compact + More Economical
- Forming Teams = More Collaborative + More Synergic

Public Transportation Networks

Social Networks

Size of a k-Truss: Definition

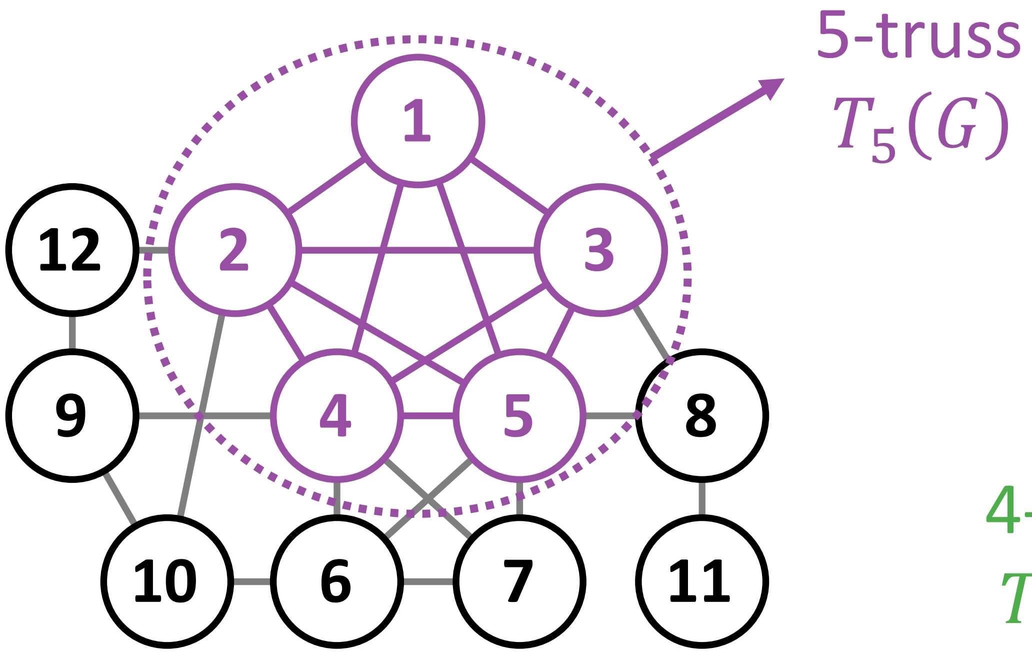
- Given $G = (V, E)$ and $k \in \mathbb{N}$
- The ***k*-truss** $T_k(G)$ of G is the maximal subgraph where each edge in $T_k(G)$ is contained in at least $k - 2$ triangles in it
  - A triangle is a complete subgraph of size three



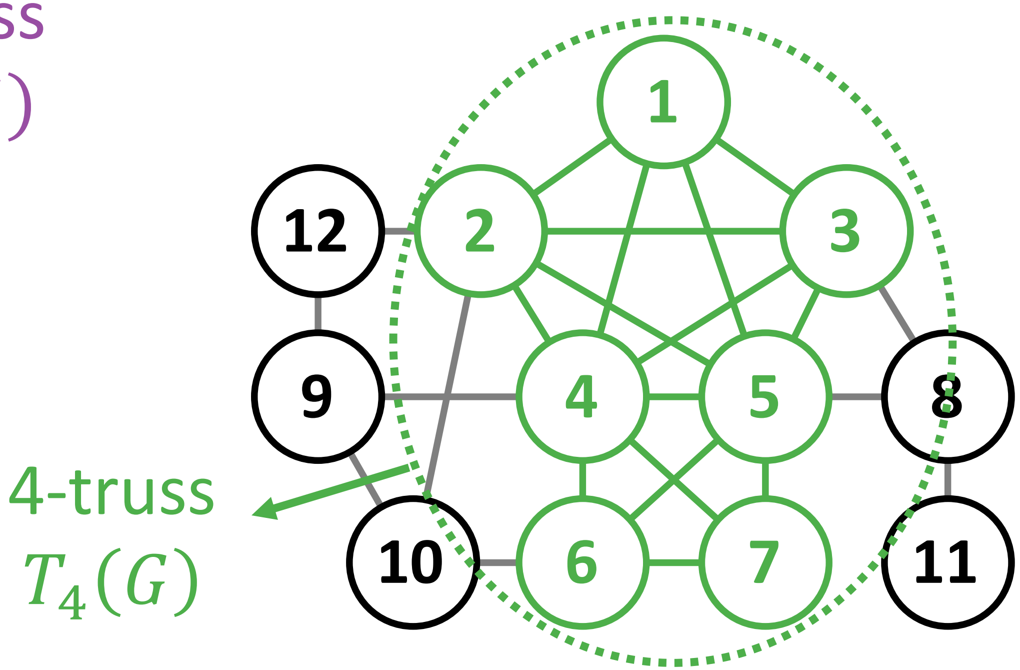
Example Graph G

Size of a k-Truss: Definition

- Given $G = (V, E)$ and $k \in \mathbb{N}$
- The ***k*-truss** $T_k(G)$ of G is the maximal subgraph where each edge in $T_k(G)$ is a part of at least $k - 2$ triangles in it



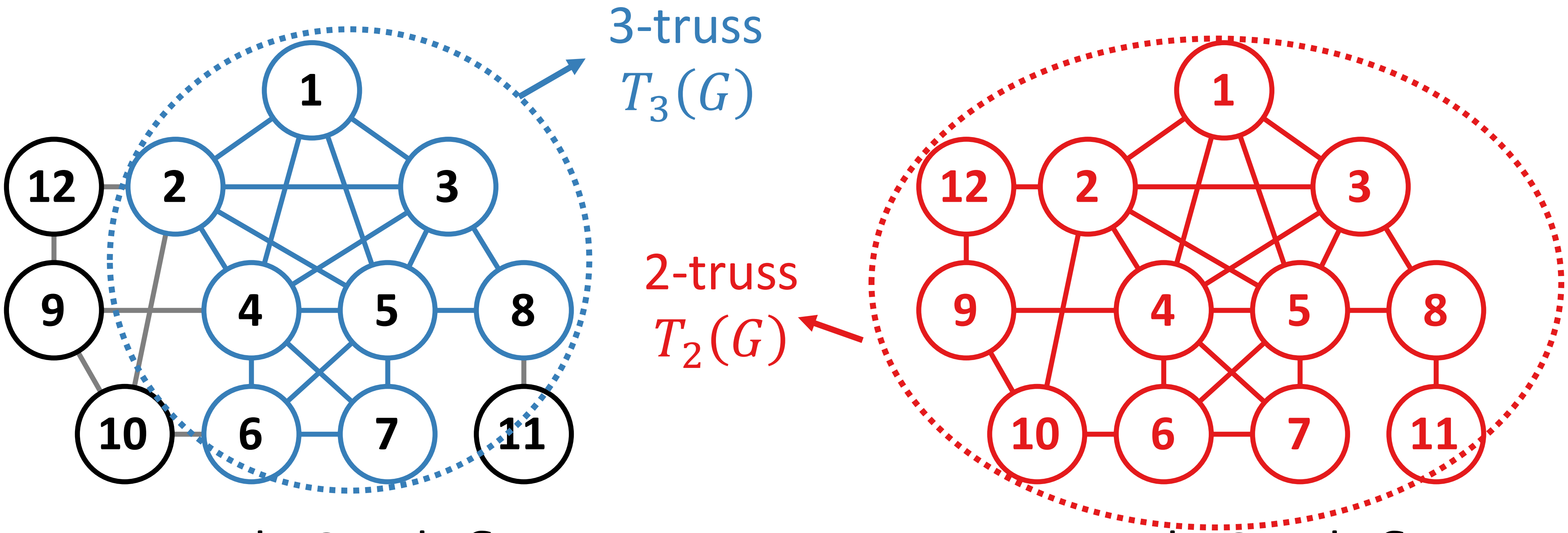
Example Graph G



Example Graph G

Size of a k-Truss: Definition

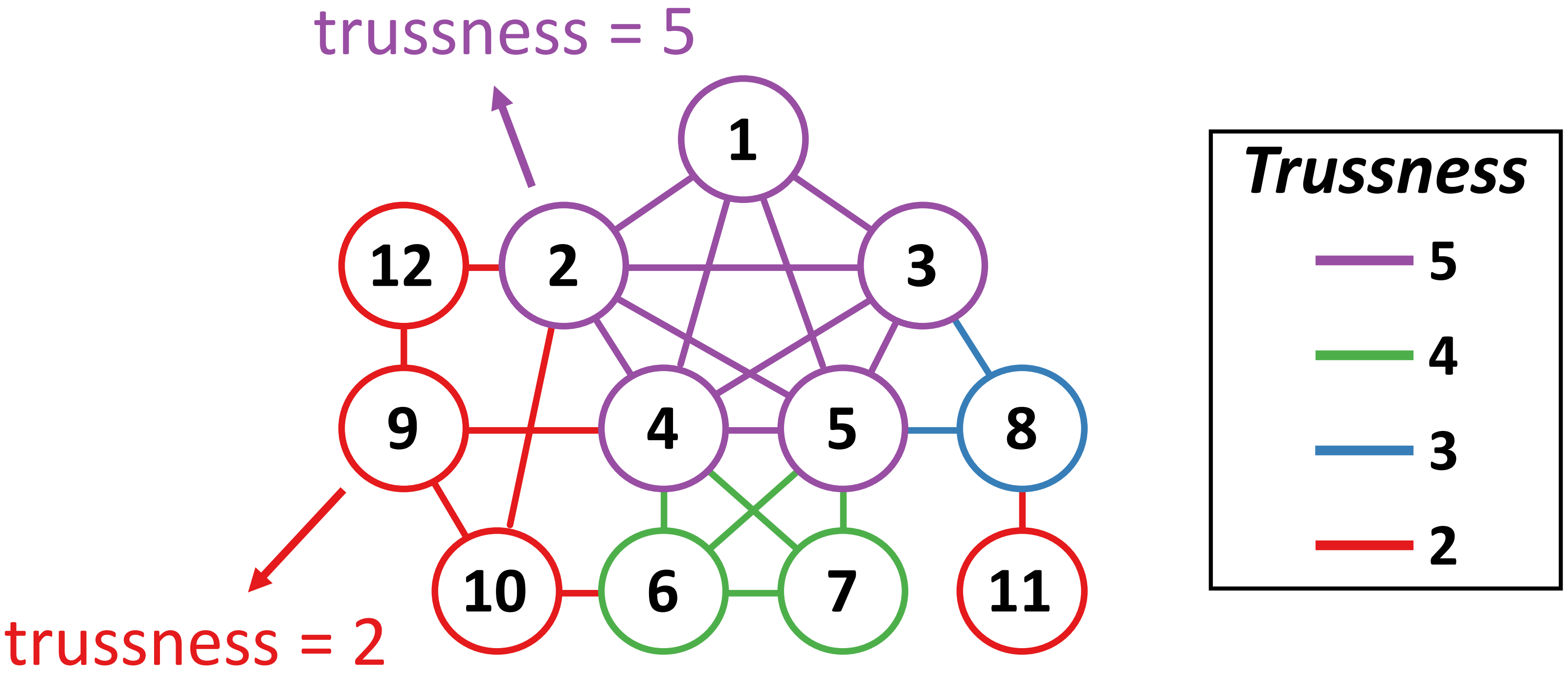
- Given $G = (V, E)$ and $k \in \mathbb{N}$
- The ***k*-truss** $T_k(G)$ of G is the maximal subgraph where each edge in $T_k(G)$ is a part of at least $k - 2$ triangles in it



Example Graph G Example Graph G

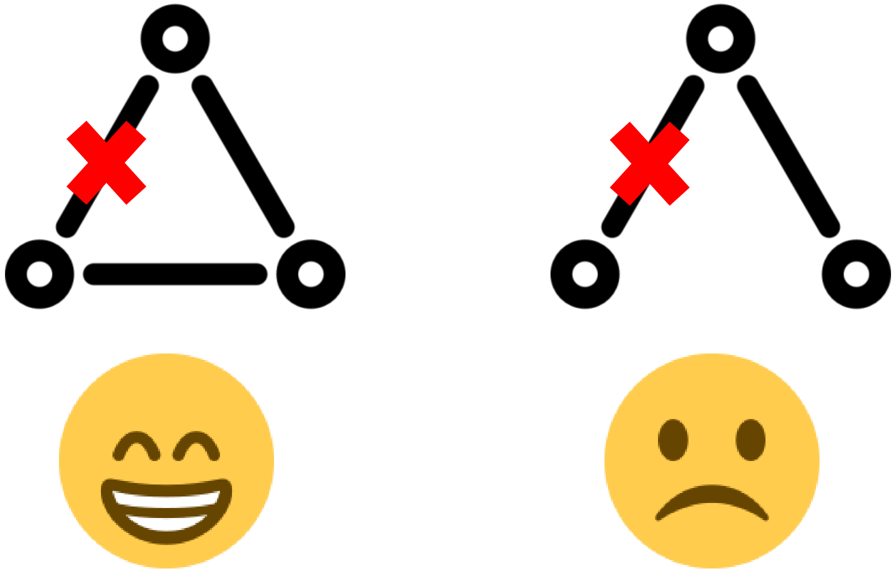
Related Concept: Trussness

- The **trussness** of an edge e is the maximum k s.t. e is in the k-truss
- The **trussness** of a node v is the maximum k s.t. v is in the k-truss



Why k-Trusses?

- Intuitively, **triangles** increase the robustness of graphs

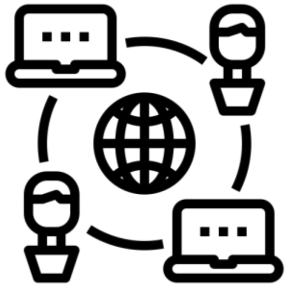


Why k-Trusses?

- Theoretically,
  - Node-level: engagement (degree) is guaranteed
  - Edge-level: interrelatedness (# triangles) is guaranteed
  - Subgraph-level: closeness (diameter) is guaranteed
- Practically meaningful

Transportation Systems

Social Networks



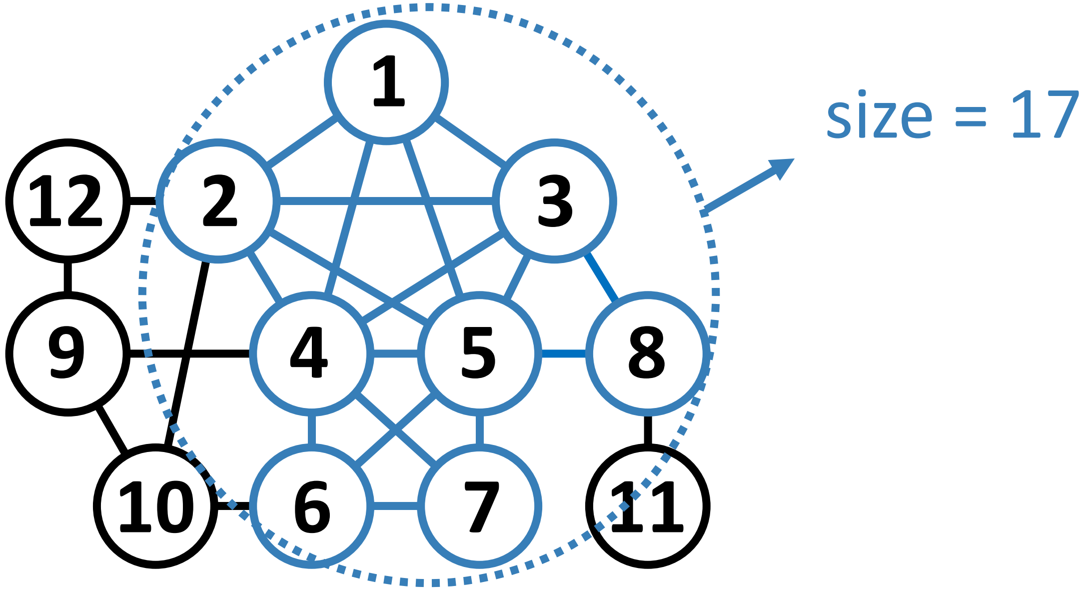
Communication Systems

Recommender Systems

Problem Statement

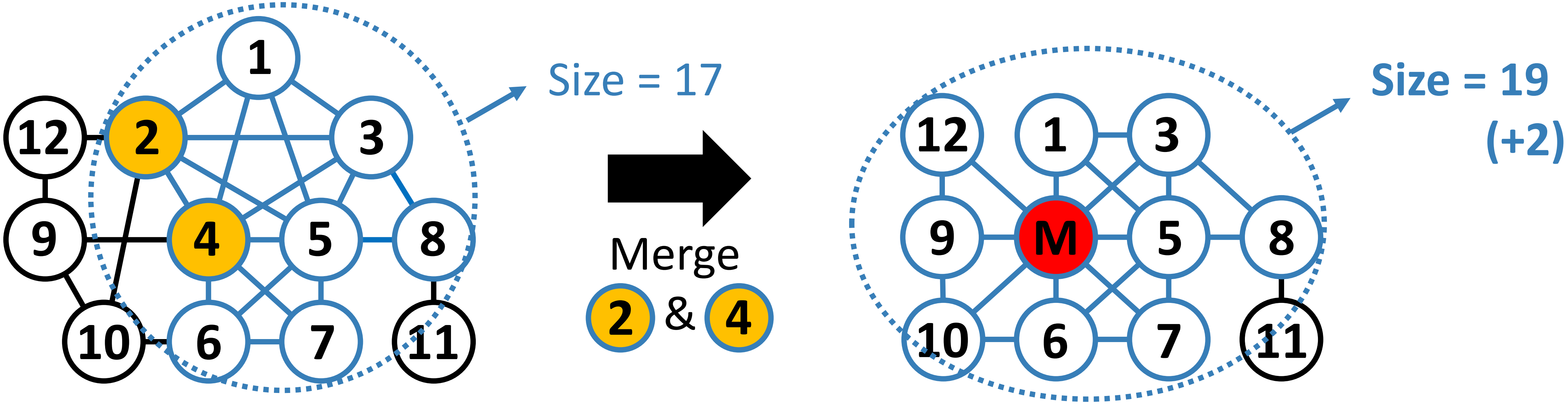
- **Given:** $G = (V, E)$, $k \in \mathbb{N}$, "budget" $b \in \mathbb{N}$
- **Find:** b pairs of nodes to be merged
- **To Maximize:** # edges in the k-truss after merging those pairs
- **Example:**

"Which node pair should we merge to maximize the size of the 3-truss?"



Problem Statement: Example

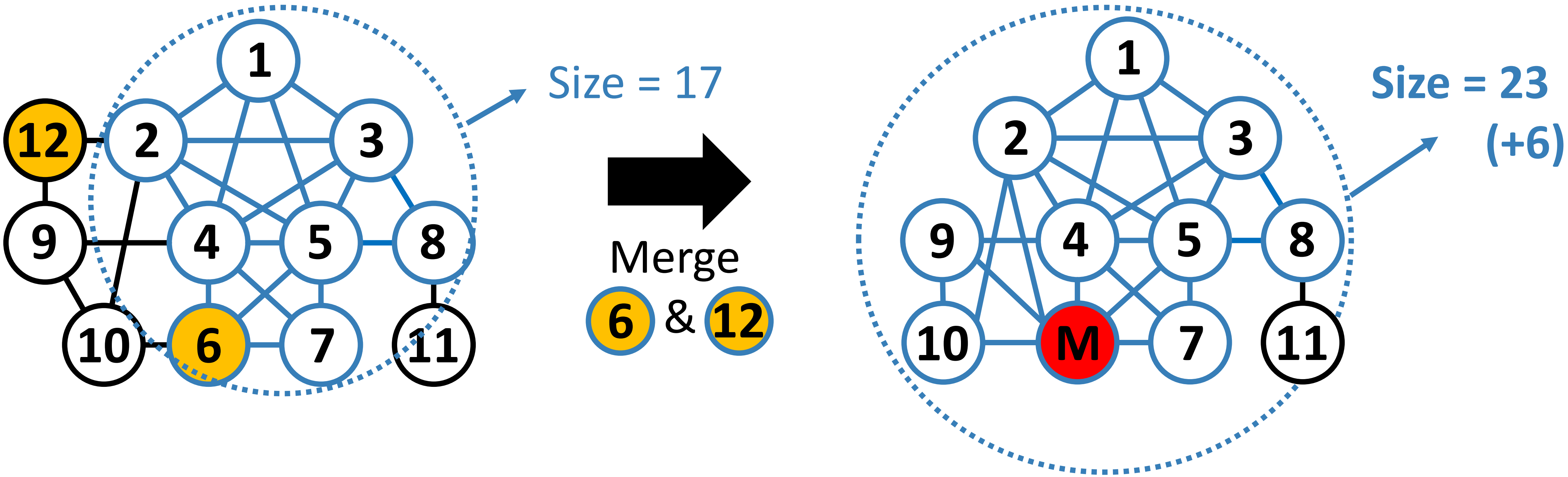
"Which node pair should we merge to maximize the size of the 3-truss?"



Merging nodes 2 and 4 increases the size by 2

Problem Statement: Example

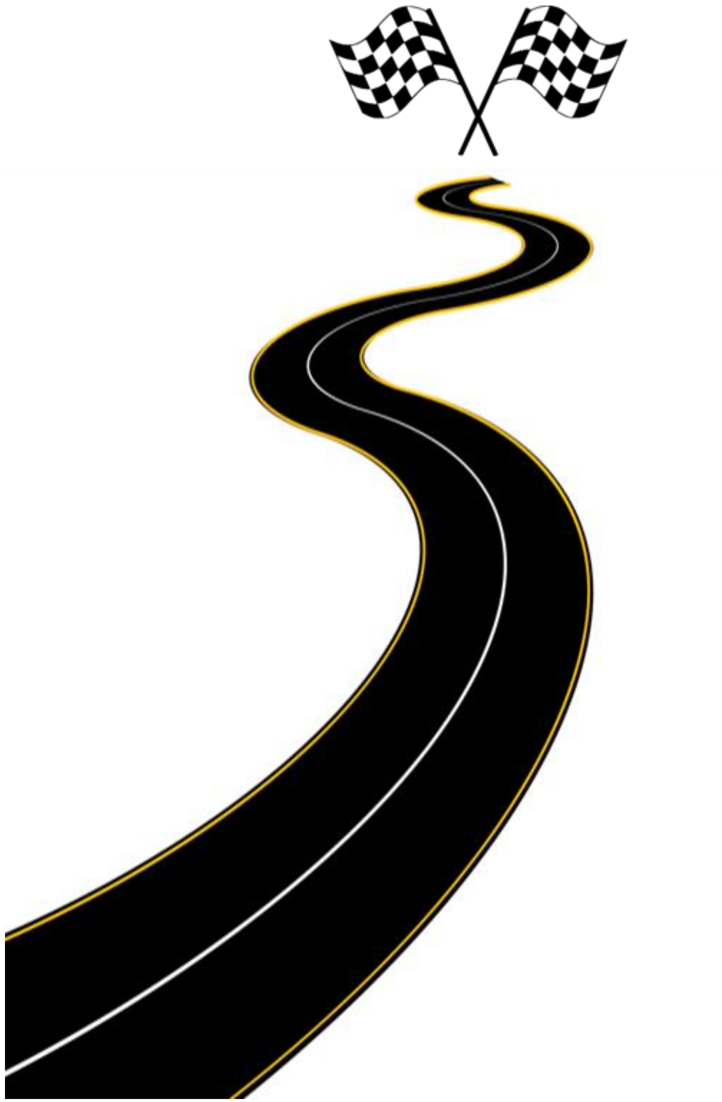
"Which node pair should we merge to maximize the size of the 3-truss?"



Merging nodes 6 and 12 increases the size by 6

Roadmap

- Formulation
- **Analysis & Algorithms <<**
- Experiments
- Conclusions



Hardness and a Naïve Algorithm

- Theorem: The problem is **NP-hard**
  - We need to find practical and efficient heuristics.

- A naïve greedy algorithm
  - Repeat until the budget is exhausted
    - For each possible merger
      - Compute # edges in the k-truss after the merger
    - Operate the best merger

- The naïve greedy algorithm takes $O(b|V|^2|E|^{1.5})$ time! 😵
  - b: budget, V: set of nodes, E: set of edges

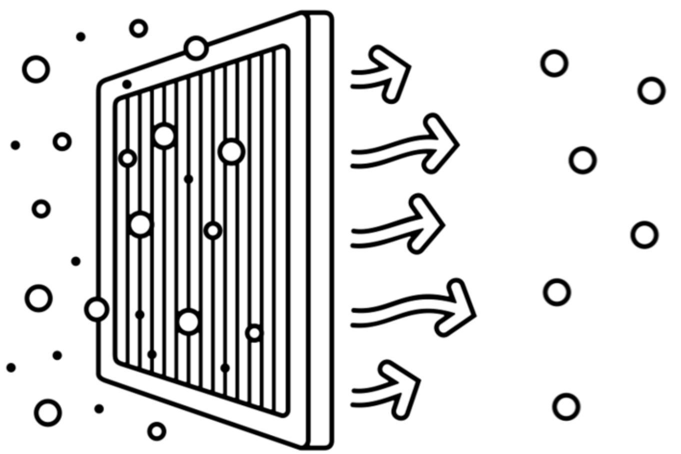
Research Questions

- The naïve greedy algorithm takes $O(b|V|^2|E|^{1.5})$ time!
  - The k-truss is computed for each of $O(|V|^2)$ possible node pairs.

Research Questions

- The naïve greedy algorithm takes $O(b|V|^2|E|^{1.5})$ time!
  - The k-truss is computed for each of $O(|V|^2)$ possible node pairs.
- Q: How can we reduce the time complexity?
- A: Rapidly filter **promising node pairs** and focus on them

All
node pairs

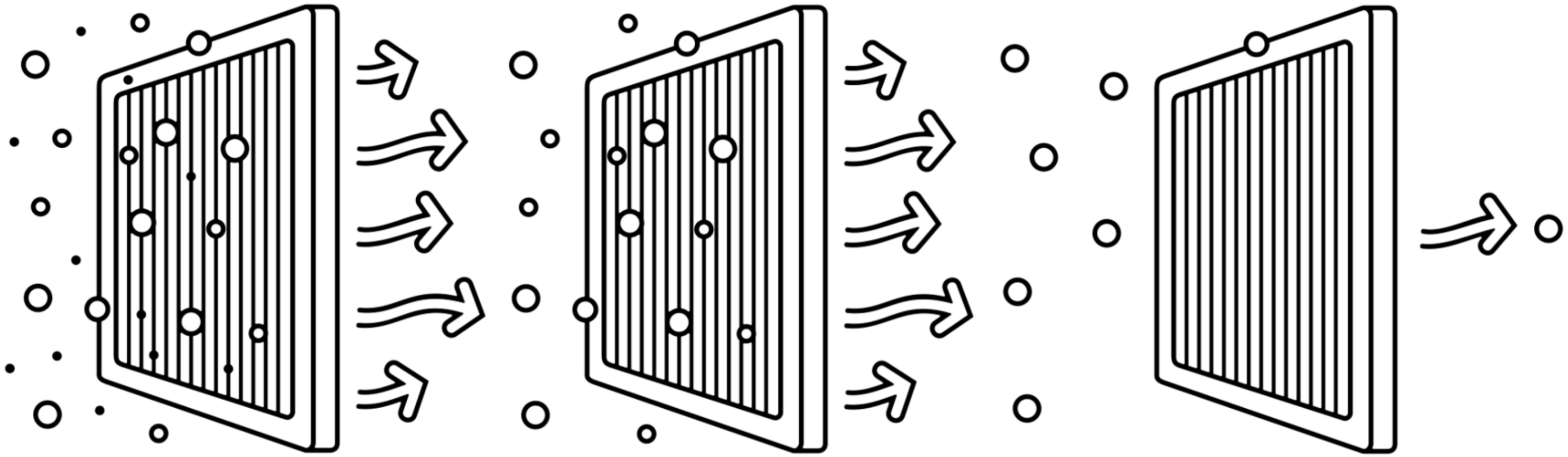


Promising
node pairs

Algorithm Overview

- Repeat until the budget is exhausted
  - **S1. Node filter:** find promising nodes
  - **S2. Pair filter:** find promising pairs of promising nodes
  - **S3. Final selector:** among the promising pairs and merge the best

All
node pairs

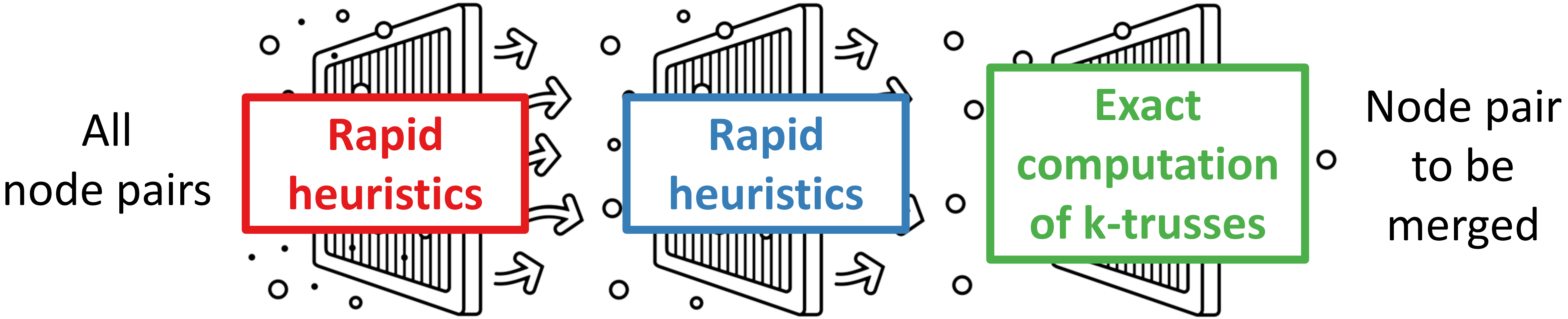


Node pair
to be
merged

S1. Node filter　　S2. Pair filter　　S3. Final selector

Algorithm Overview

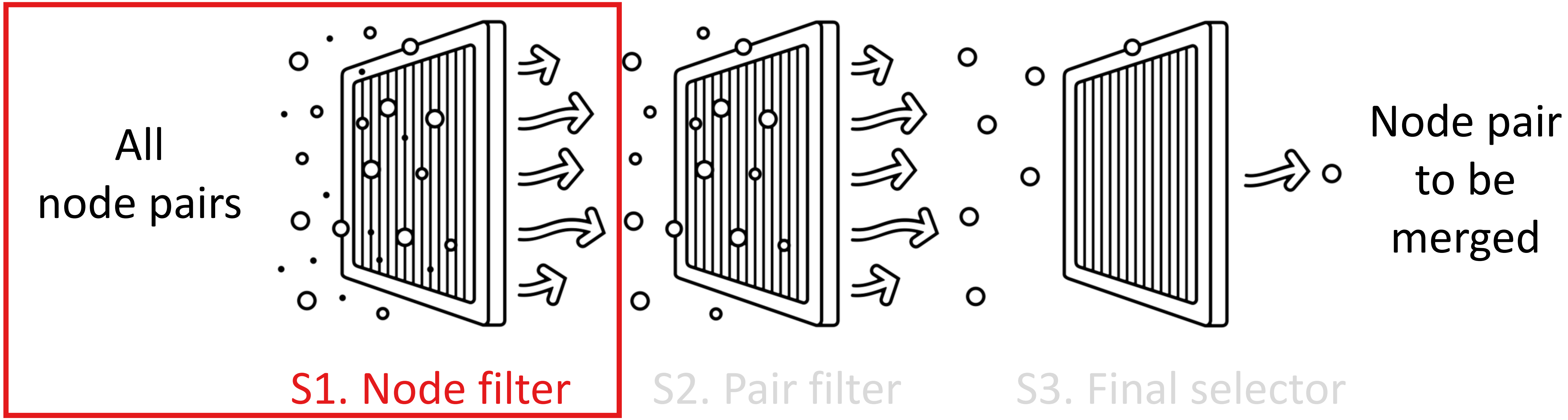
- Repeat until the budget is exhausted
  - **S1. Node filter:** find promising nodes
  - **S2. Pair filter:** find promising pairs of promising nodes
  - **S3. Final selector:** among the promising pairs and merge the best



S1. Node filter S2. Pair filter S3. Final selector

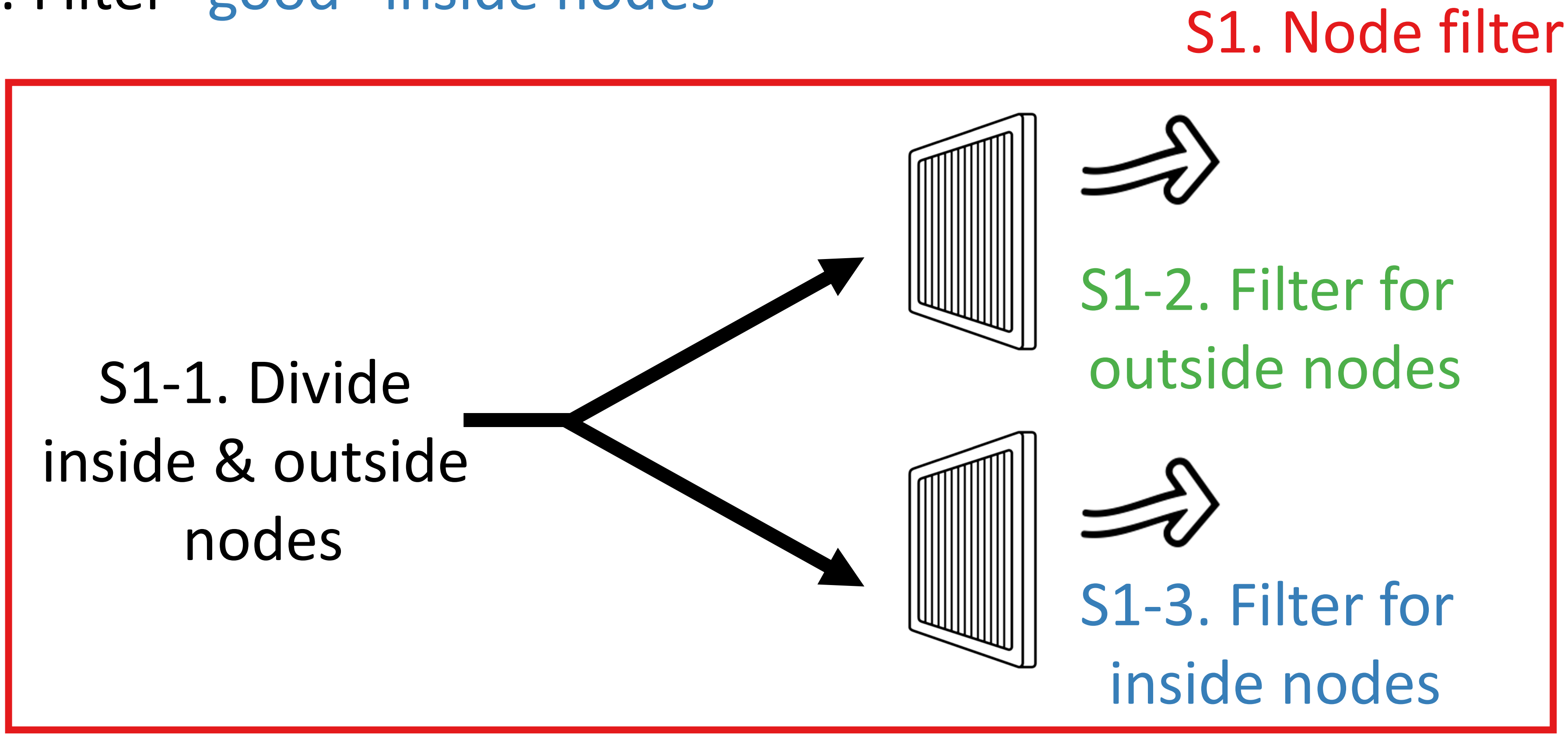
Algorithm Overview

- Repeat until the budget is exhausted
  - **S1. Node filter:** find promising nodes <<
  - S2. Pair filter: find promising pairs of promising nodes
  - S3. Best selector: evaluate the promising pairs and merge the best



Step 1. Node Filter

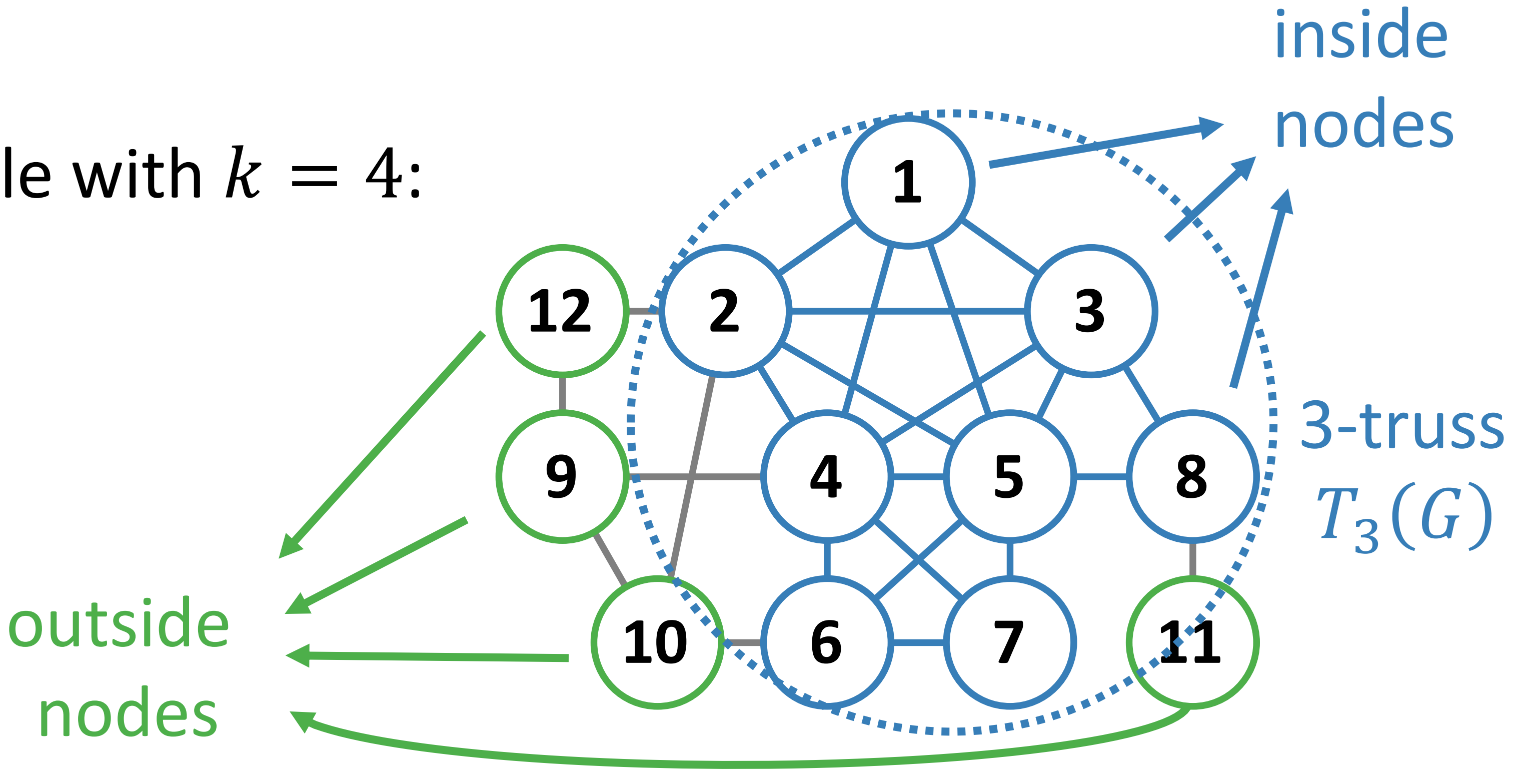
- S1-1. Divide the inside nodes and outside nodes
- S1-2. Filter "good" outside nodes
- S1-3. Filter "good" inside nodes



Step 1-1. Divide the Inside and Outside Nodes

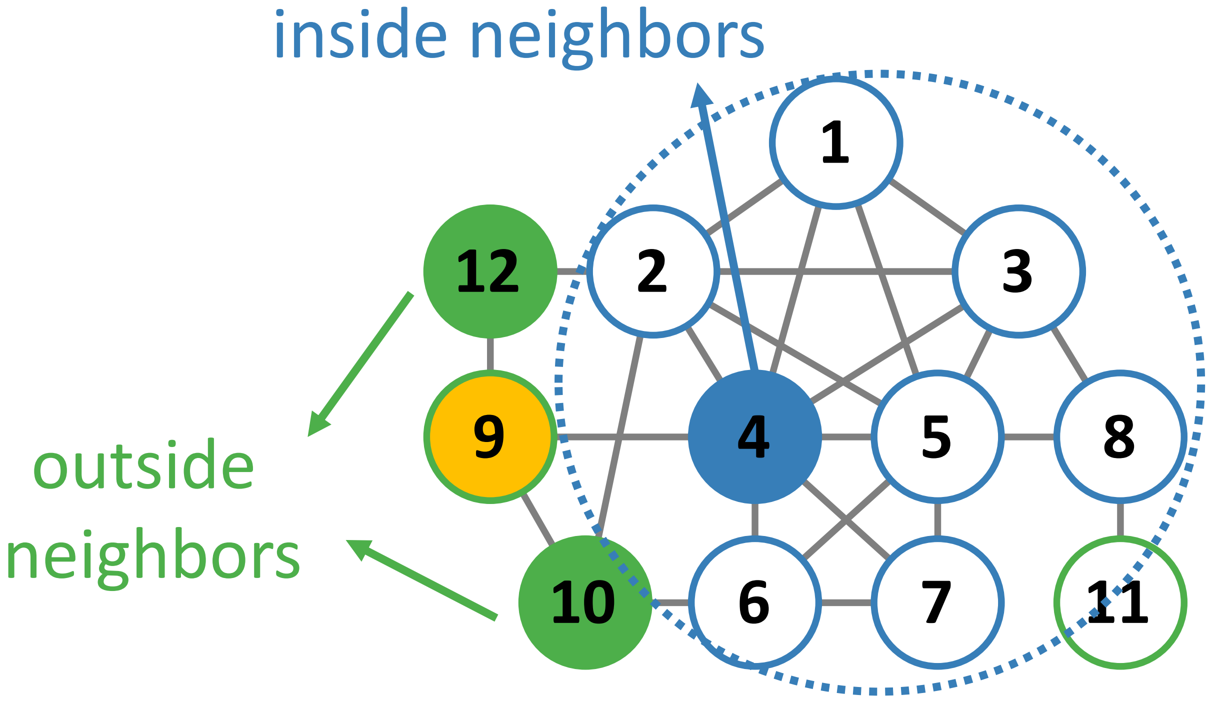
- Given a graph $G = (V, E)$ and $k \in \mathbb{N}$
- A node $v \in V$ is if an **inside node** if its trussness $t(v) \geq k - 1$
- Otherwise, it is an **outside node**

- Example with $k = 4$:



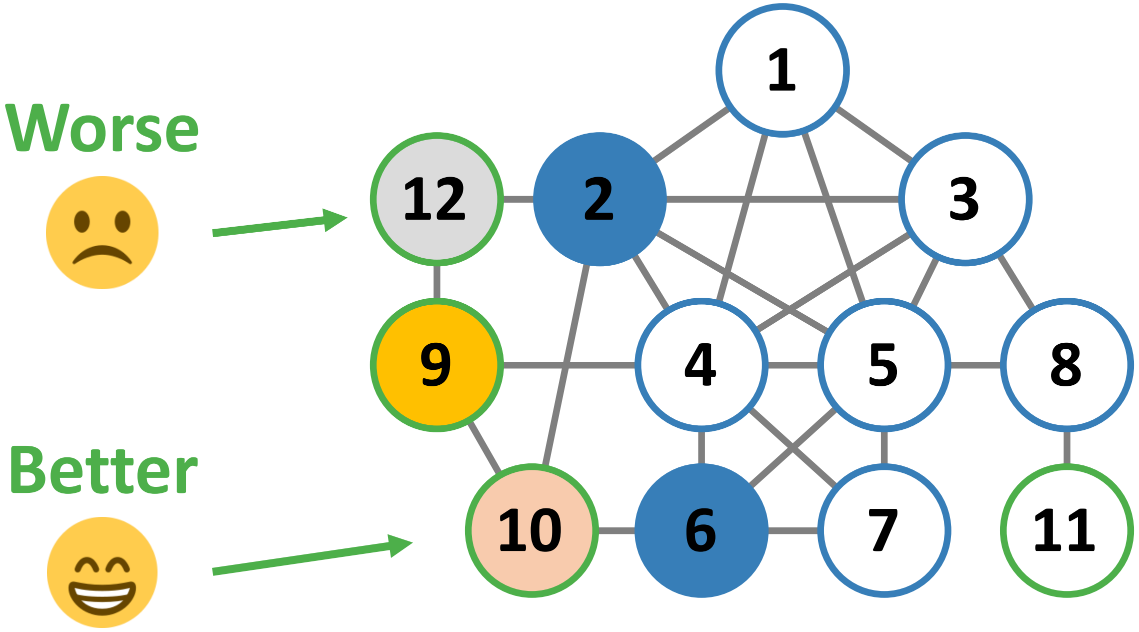
Step 1-1. Divide the Inside and Outside Nodes

- Given a **node u** $\in V$
- The **inside neighbors** of u are the neighbors of u that are inside nodes
- The **outside neighbors** of u are the neighbors of u that are outside nodes



Step 1-2. Filter Good Outside Nodes

- Lemma:
  - *For two outside nodes a, b and any c,*
  - *if the inside neighborhood of a is a strict superset of that of b,*
  - *then merging a and c is always better than merging b and c*
- Example:



Step 1-2. Filter Good Outside Nodes

- Implications:
  - Outside nodes with many inside neighbors are good!
  - We only need to consider those with **maximal inside neighborhood**
- Problem formulation – Maximal-Set Enumeration (Yellin, 1992):
  - **Given:** a set of sets (i.e., sets of inside neighbors of outside nodes)
  - **Find:** maximal sets
- Algorithm:
  - We use an existing algorithm that is simple yet effective

Step 1-3. Filter Good Inside Nodes

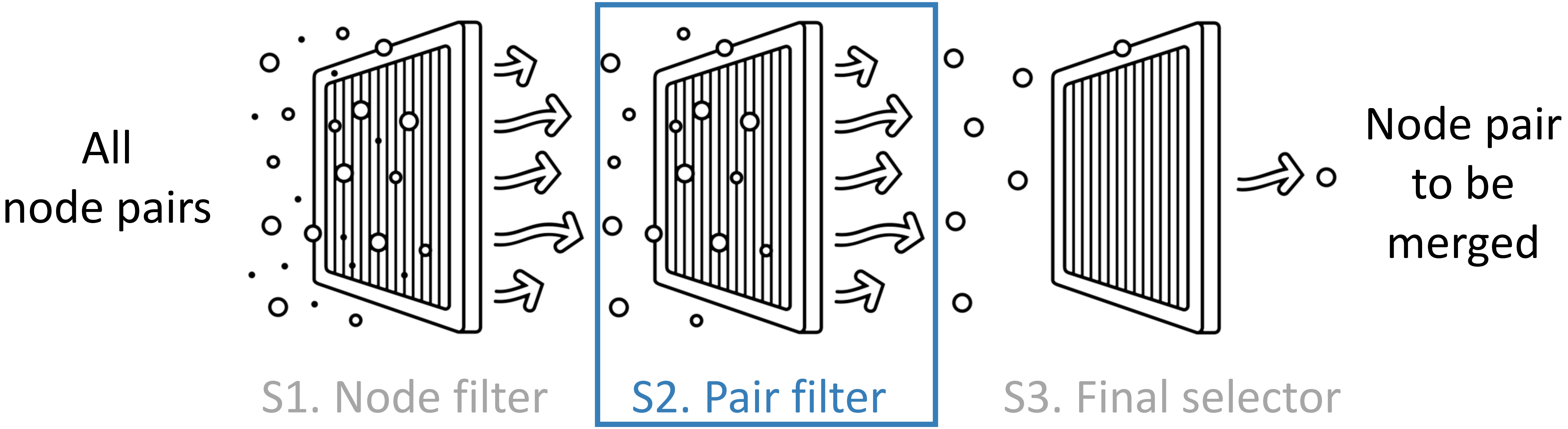
- For each inside node v,
- **Count the number of promising neighbors**, which can potentially enter the k-truss after v is merged with another node
- Technically, the **promising neighbors** of an inside node v are the neighbors of v with trussness $k-1$
- Then, **choose the top-n** inside nodes based on the number

Estimate potential

Rank nodes

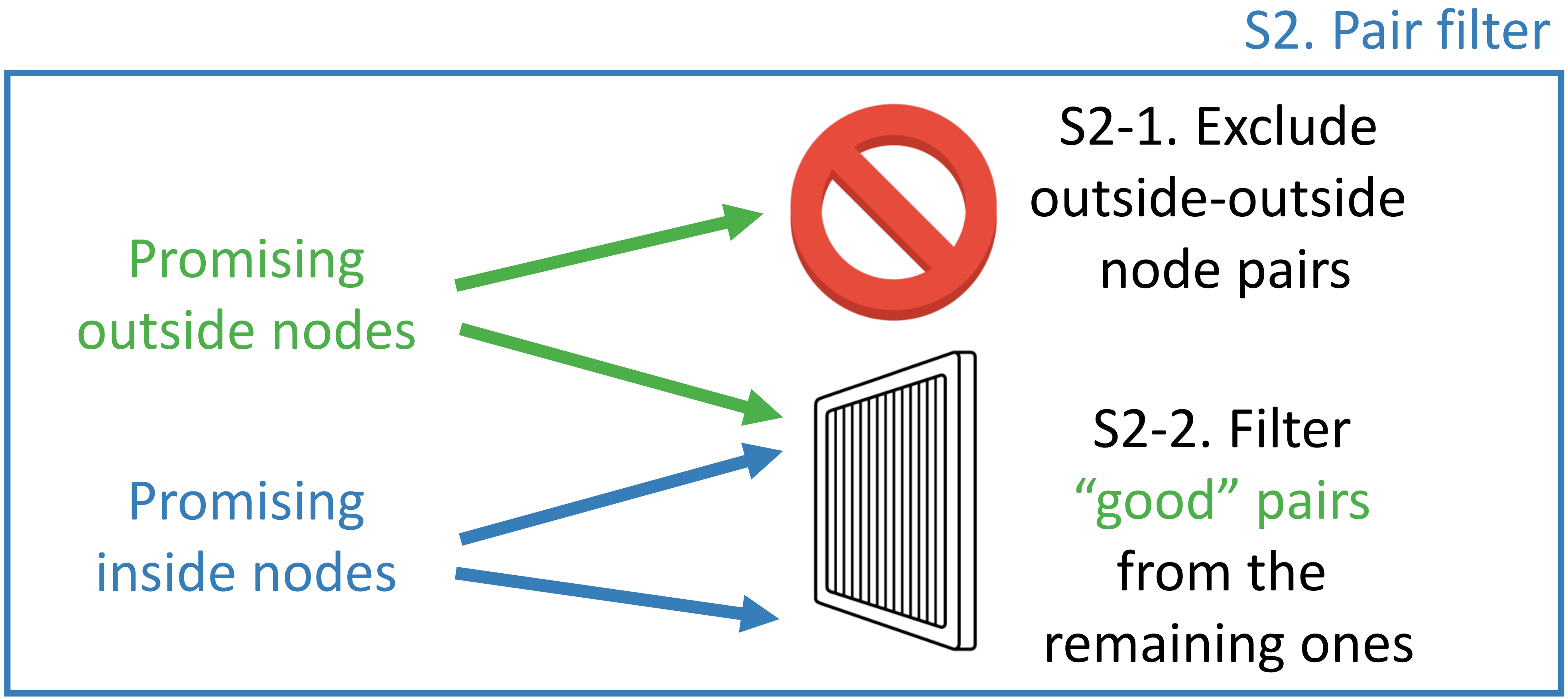
Algorithm Overview

- Repeat until the budget is exhausted
  - S1. Node filter: find promising nodes
  - **S2. Pair filter:** find promising pairs of promising nodes <<
  - S3. Final selector: among the promising pairs and merge the best



Step 2. Pair Filter

- S2-1. Exclude outside-outside node pairs
- S2-2. Filter "good" pairs from the remaining ones



Step 2-1. Exclude Outside-Outside Node Pairs

- We exclude outside-outside node pairs
  - Theoretically, each edge incident to such merged nodes is not helpful
  - Empirically, there are too many such pairs
  - Thus, finding promising ones among them is costly

A large number of outside-outside node pairs that are theoretically less useful

Few good pairs that are costly to find

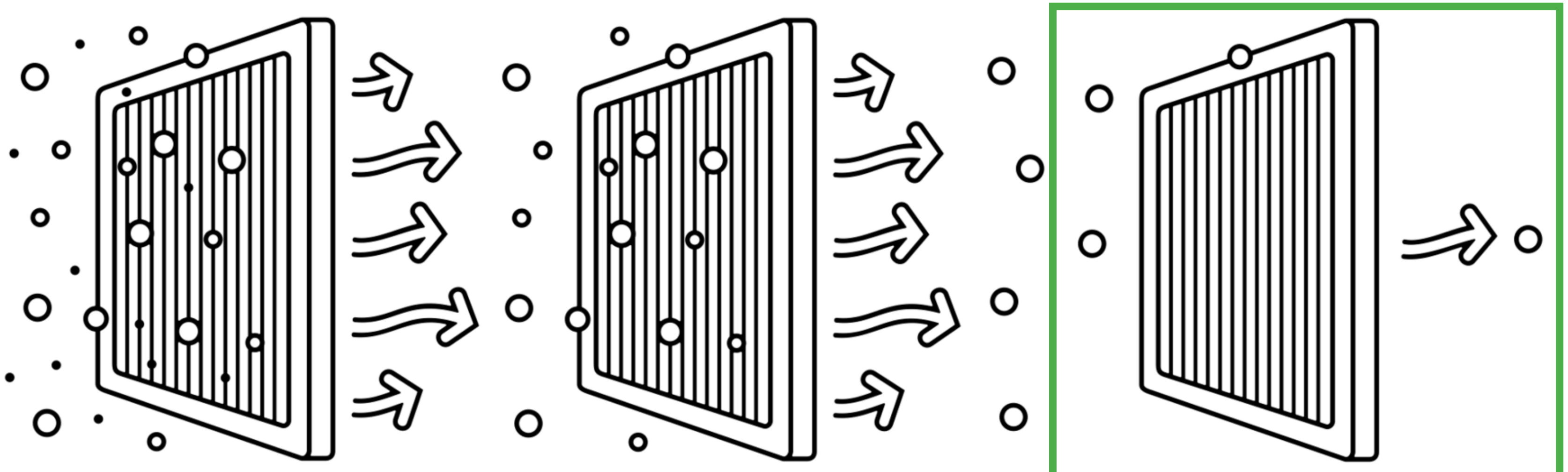
Step 2-2. Filter "Good" Pairs from the Remaining Ones

- **Score the other pairs** based on positive and negative factors
  - Inside-inside promising node pairs
  - Inside-outside promising node pairs
- **Positive factors (+1 for each)**
  - Support gains: edges in new triangles formed by merging the pair
- **Negative factors (-1 for each)**
  - Support losses: edges in triangles destroyed by merging the pair
  - Collisions: edges merged into one after merging the pair
- **Choose the top-n pairs** with the highest scores

Algorithm Overview

- Repeat until the budget is exhausted
  - S1. Node filter: find promising nodes
  - S2. Pair filter: find promising pairs of promising nodes
  - **S3. Final selector:** among the promising pairs and merge the best <<

All node pairs



Node pair to be merged

S1. Node filter

S2. Pair filter

S3. Final selector

Step 3. Final Selector

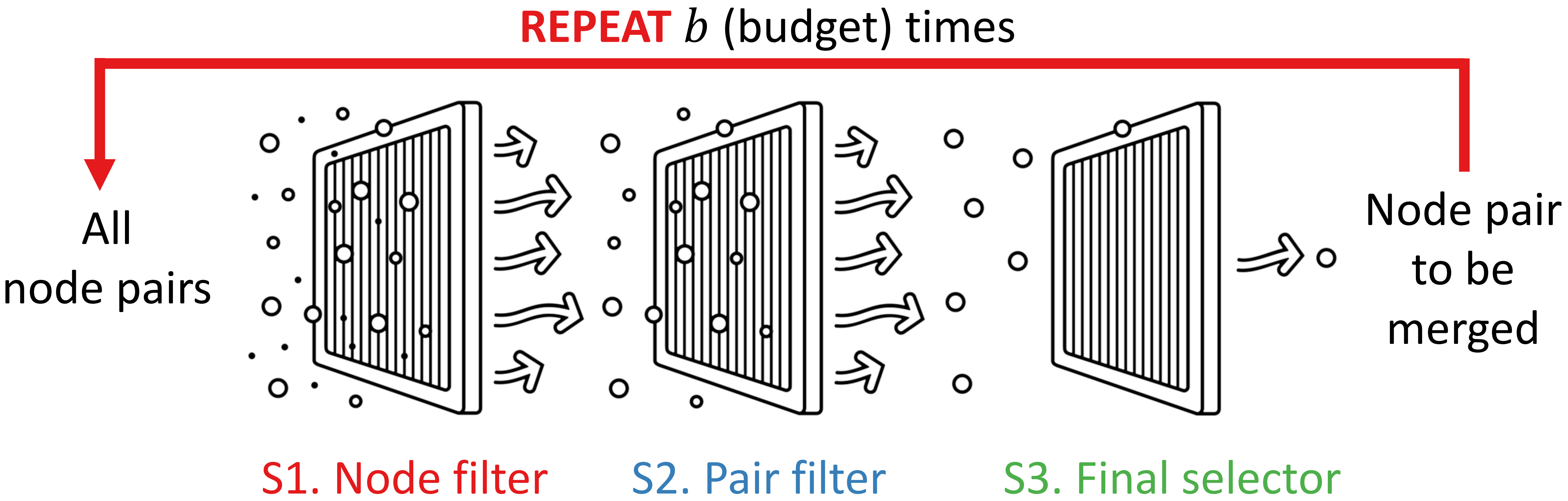
- Finally choose a node pair and merge them

- Algorithm:
  - For each **promising pair**
    - Compute the **exact gain in the objective** after their merger
  - Merge the best pair with the largest gain

- This is the only step where exact trussness is computed in $O(|E|^{1.5})$

Final Algorithm: BATMAN

- **BATMAN**: **B**est-merger se**A**rcher for **T**russ **MA**ximizatio**N**

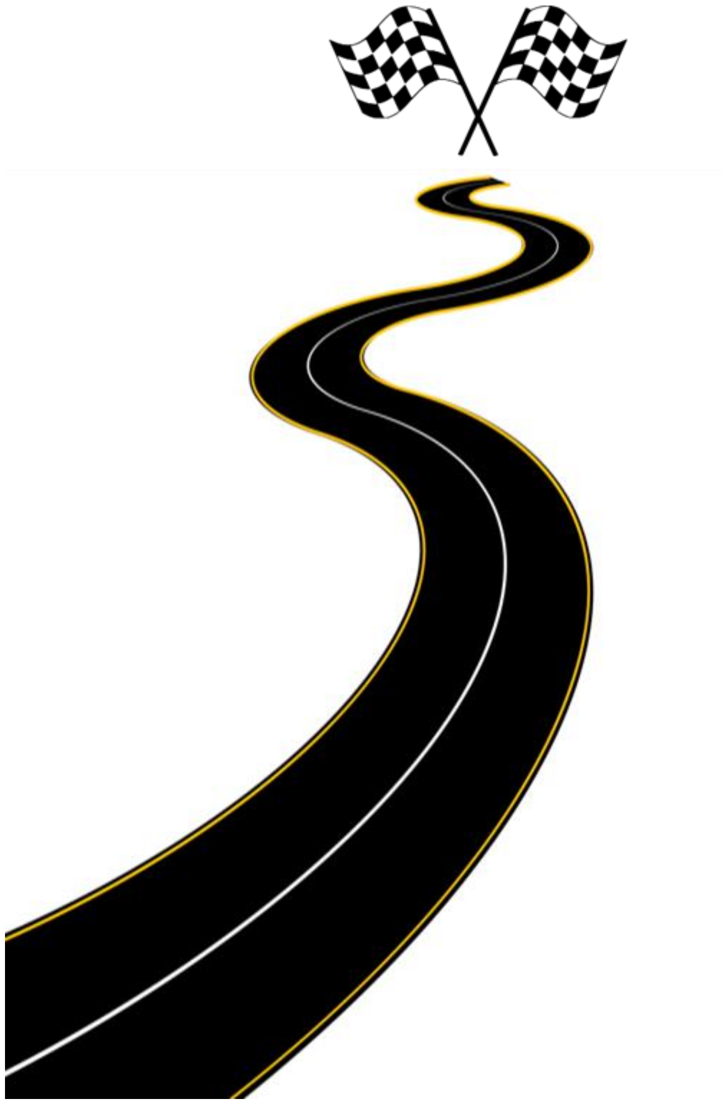


Theoretical Properties of BATMAN

- BATMAN takes $O(b|E|^{1.5} + |V|)$ time
  - Some user parameters are treated as constants
  - Recall that the naïve greedy algorithm takes $O(b|V|^2|E|^{1.5})$ time
- Unfortunately, no accuracy guarantee has been proven for BATMAN

Roadmap

- Formulation
- Analysis & Algorithms
- **Experiments <<**
- Conclusions



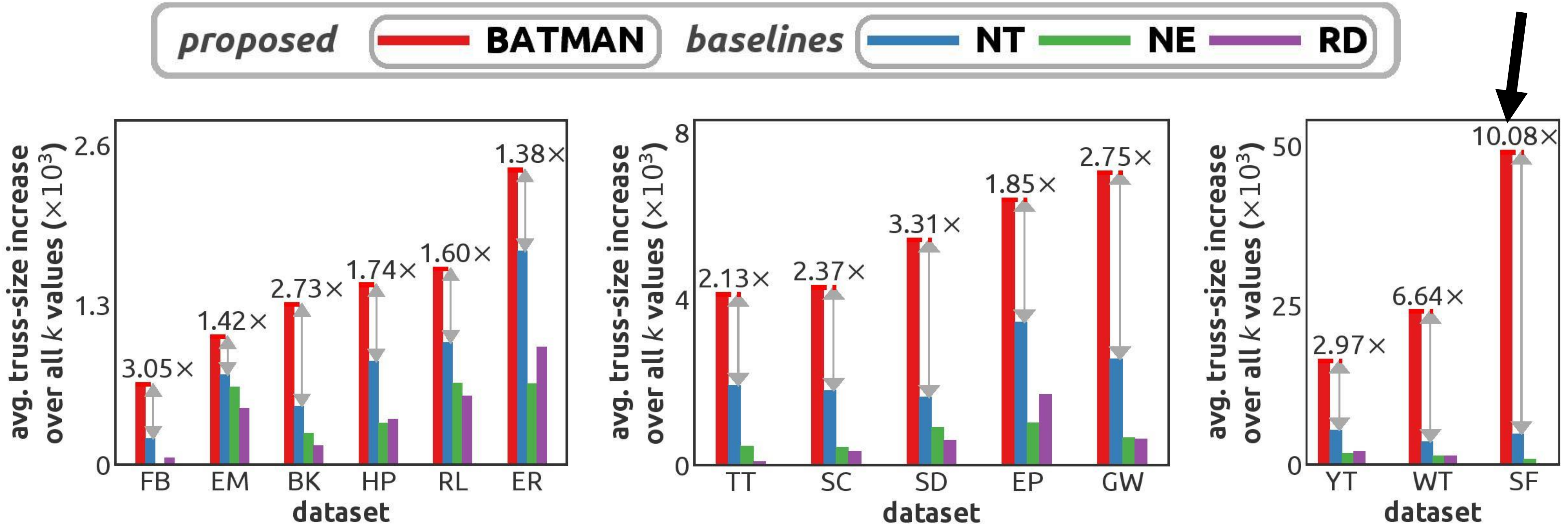
Experimental Settings

- Datasets: 14 real-worlds graphs
  - # nodes: 986 – 2.4M
  - # edges: 16k – 4.7M
- Different k @ k-trusses: 5, 10, 15, 20
- Baseline methods: different heuristics
  - Most New Triangles (**NT**): choose the edge increasing the number of triangles among the nodes in the $(k-1)$-truss
  - Most New Edges (**NE**): choose the edge increasing the number of edges between the nodes in the $(k-1)$-truss
  - RanDom (**RD**): uniform random sampling among all IIMs and IOMs
  - The naïve greedy algorithm runs out of time in all the datasets!

Effectiveness: Different Datasets

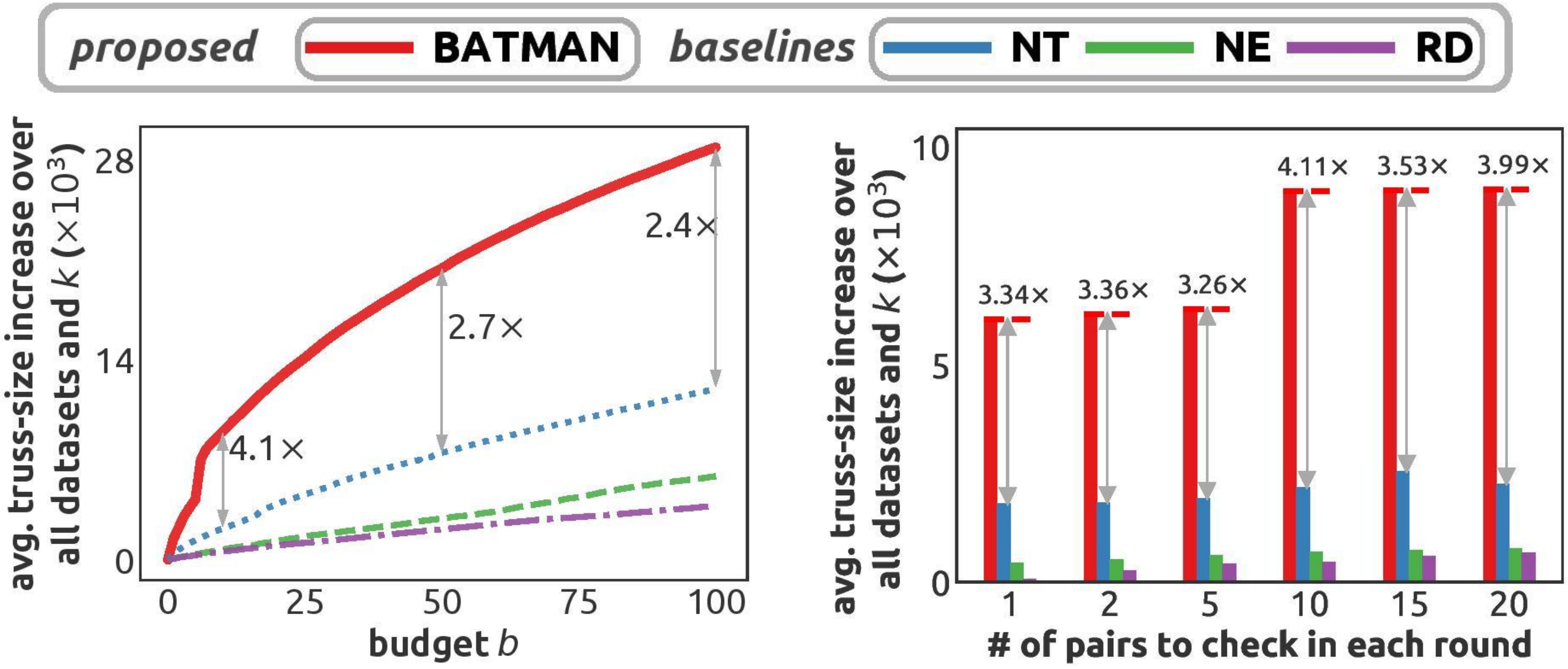
- **BATMAN** shows **consistent superiority** on different datasets
  - 1.4 - 10× better than the second-best one

∼10X Better



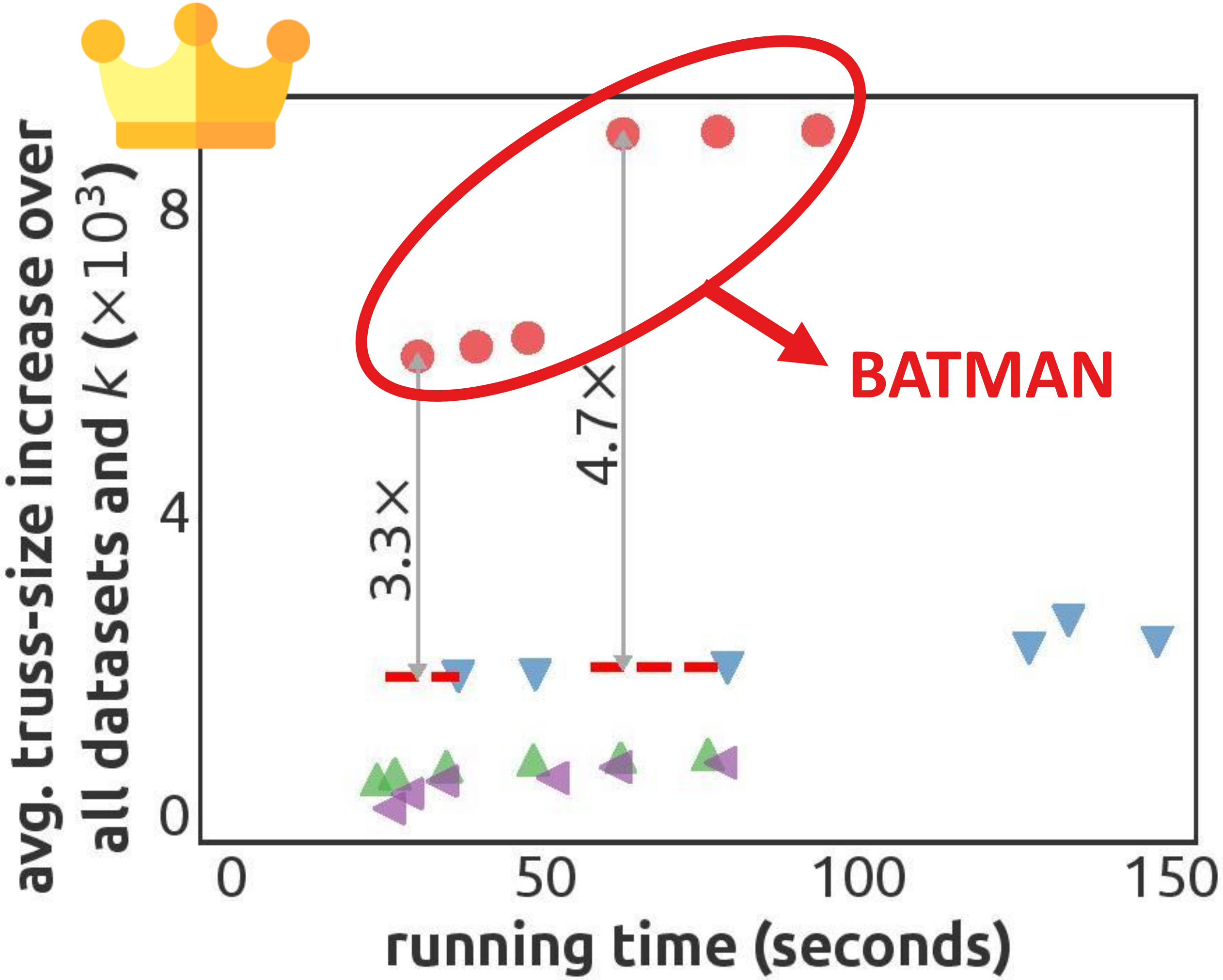
Effectiveness: Different Parameters

- **BATMAN** shows **consistent superiority** under various parameter settings



Efficiency: Speed & Performance Tradeoff

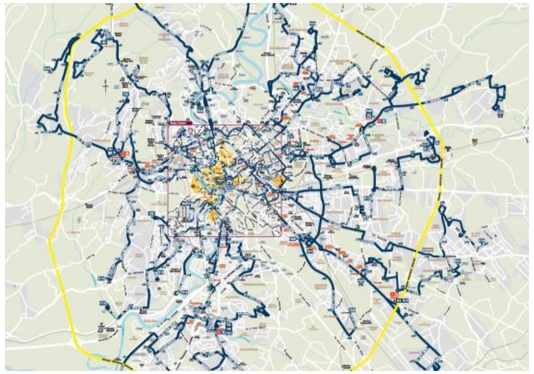
- **BATMAN** provides the best trade-off between speed & performance



Application: Merging Bus Stations

- **Datasets:** 21 bus station datasets in different cities
- **Goal:** Merging stations to improve the robustness of bus network
- **Constraints:** Only stations close enough can be merged
  - But there are still many possible choices!
- **Baseline methods:**
  - **BATMAN (BM):** Try to enlarge a k-truss
  - **Core (CR)**: Try to enlarge a k-core
  - **Constraints (CS):** Randomly pick station pairs satisfying the constraints
  - **Closest (CL):** Pick the closest station pairs

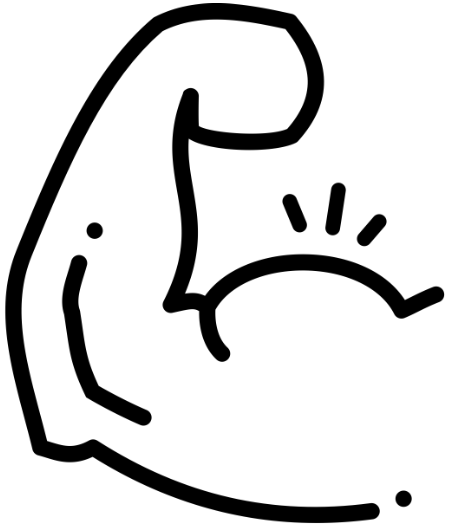
Paris



Rome

Application: Merging Bus Stations

- **Metrics:** **8 robustness measures** for transportation networks
  - **VB:** Average vertex betweenness
  - **EB:** Average edge betweenness
  - **ER:** Effective resistance
  - **SG:** Spectral gap
  - **NC:** Natural connectivity
  - **AD:** Average distance
  - **TS:** Transitivity
  - **LC:** Average local clustering coefficient



- All these measures have been used in existing transportation network literature

Application: Merging Bus Stations

- BATMAN (BM) shows the largest improvement in robustness overall
  - Each robustness metric is computed after merging the stations
  - For each metric, the Z-score and the average ranking are computed

metric	BM	CR	CS	CL
VB	2.0952	**2.0476**	2.0952	3.7619
EB	2.0476	**2.0000**	2.1429	3.8095
ER	3.2857	**1.6190**	1.8095	3.2857
SG	**1.7619**	2.5238	2.3810	2.6667
NC	**1.0000**	2.3333	2.7619	2.9048
AD	2.6190	2.2381	**1.2381**	3.9048
TS	**1.0000**	3.3810	2.2381	3.3810
LC	**1.5714**	3.2381	1.6190	3.5238
average	**1.9226**	2.4226	2.0357	3.4048

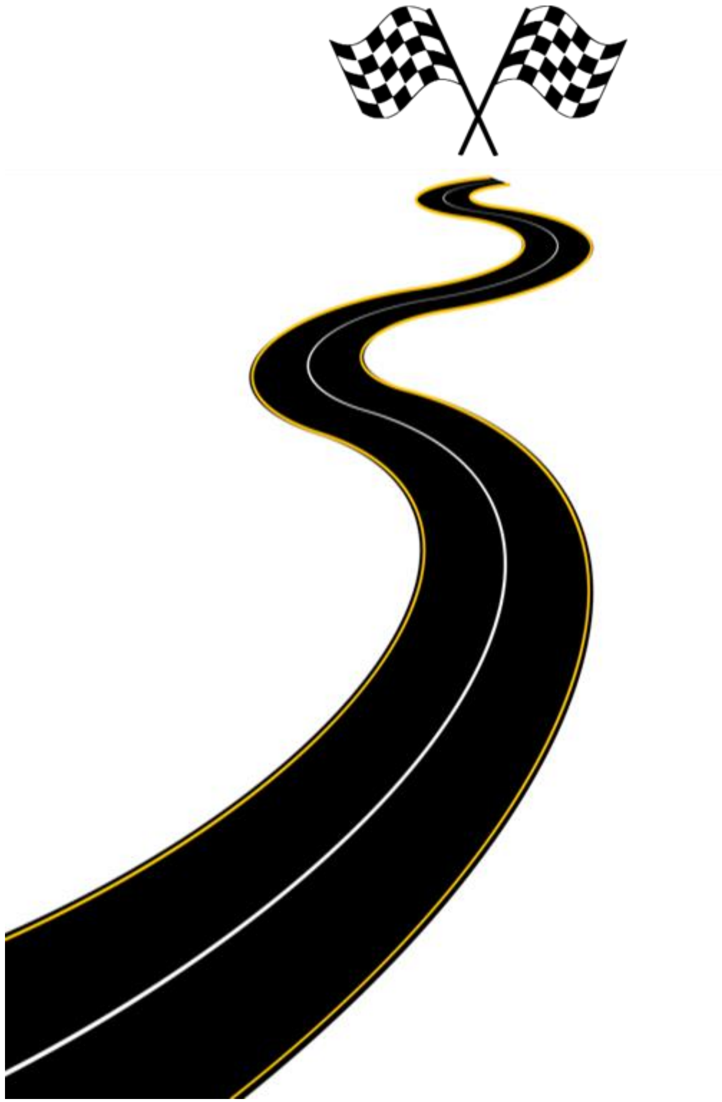
Z-scores (higher = better)

metric	BM	CR	CS	CL
VB	2.0952	**2.0476**	2.0952	3.7619
EB	2.0476	**2.0000**	2.1429	3.8095
ER	3.2857	**1.6190**	1.8095	3.2857
SG	**1.7619**	2.5238	2.3810	2.6667
NC	**1.0000**	2.3333	2.7619	2.9048
AD	2.6190	2.2381	**1.2381**	3.9048
TS	**1.0000**	3.3810	2.2381	3.3810
LC	**1.5714**	3.2381	1.6190	3.5238
average	**1.9226**	2.4226	2.0357	3.4048

Ranking (lower = better)

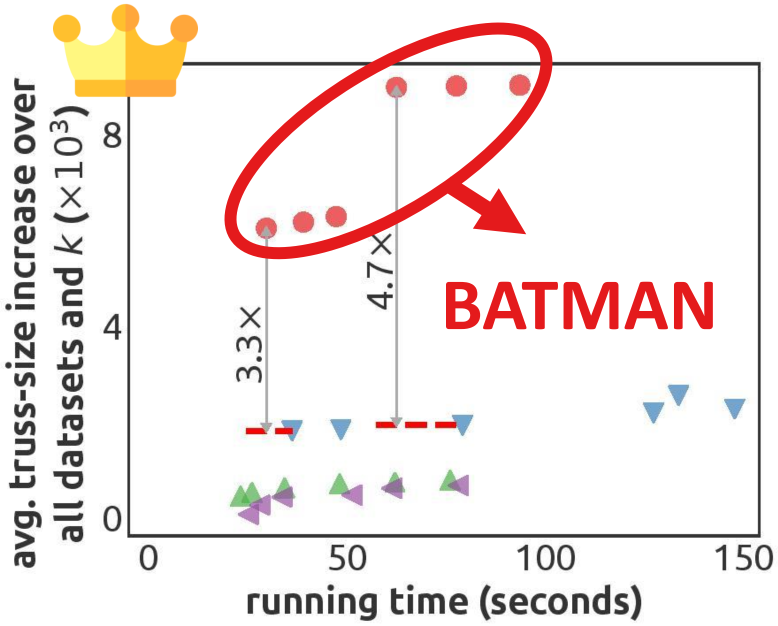
Roadmap

- Formulation
- Analysis & Algorithms
- Experiments
- **Conclusions <<**



Conclusions

- **Novel Problem** of improving graph cohesiveness by merging nodes
- **Theoretical Analysis** including the NP-hardness of the problem
- **Fast and Effective Algorithm** based on the analysis
- **Empirical Validation** including an application to real-world scenarios



Code: bit.ly/truss_merge_code

Image attribution: SEOULMETRO, Freepik, R Diepenheim, Oleksandr Panasovesky, macrovector, rawpixel.com, Sentavio, Harryarts, Cornecoba, pch.vector, brgfx, Webtechops LLP