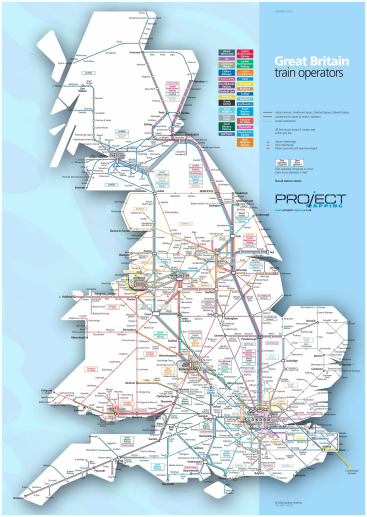
Inclusive Spanning Graphs

Iain Dixon, Vlad González-Zelaya, Julián Salas, and Daniel Archambault

Tenth Annual UK System Research Challenges Workshop
21–23 April 2026

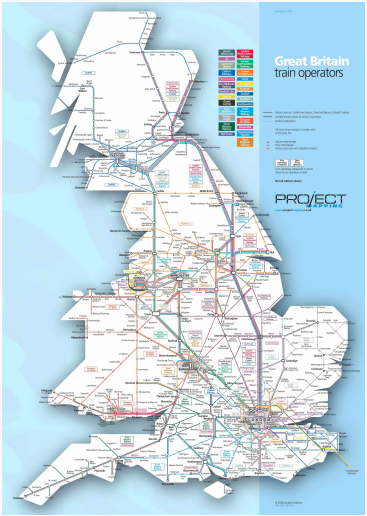
Inclusive Spanning Graphs

- **Minimal** subgraphs preserving *connectivity*
- May be prioritised for *efficiency* or *security*
- **Diverse** spanning graphs (SGs) prevent *full* network collapse by a single party
- **Representative** SGs ensure a fair share of the priority network among parties
- **Robust** SGs prevent even *partial* shutdown of the priority network by a single party



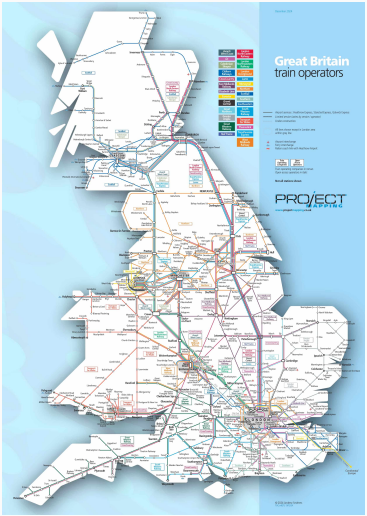
Inclusive Spanning Graphs

- **Minimal** subgraphs preserving *connectivity*
- May be prioritised for *efficiency* or *security*
- **Diverse** spanning graphs (SGs) prevent *full* network collapse by a single party
- **Representative** SGs ensure a fair share of the priority network among parties
- **Robust** SGs prevent even *partial* shutdown of the priority network by a single party



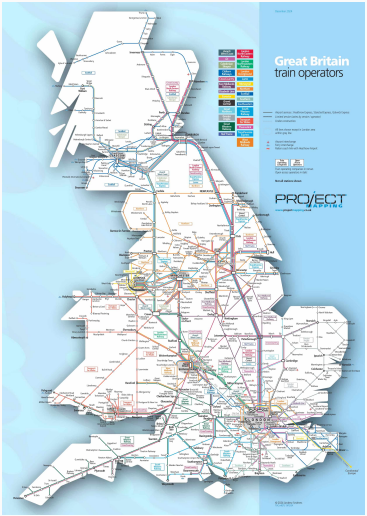
Inclusive Spanning Graphs

- **Minimal** subgraphs preserving *connectivity*
- May be prioritised for *efficiency* or *security*
- **Diverse** spanning graphs (SGs) prevent *full* network collapse by a single party
- **Representative** SGs ensure a fair share of the priority network among parties
- **Robust** SGs prevent even *partial* shutdown of the priority network by a single party



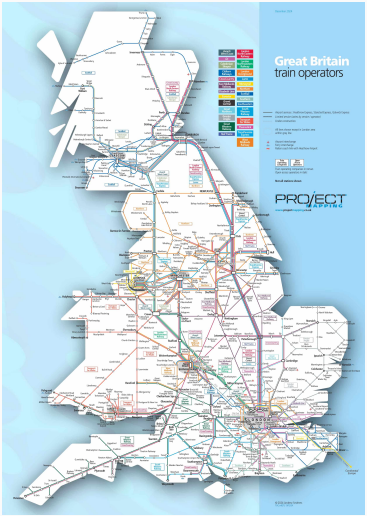
Inclusive Spanning Graphs

- **Minimal** subgraphs preserving *connectivity*
- May be prioritised for *efficiency* or *security*
- **Diverse** spanning graphs (SGs) prevent *full* network collapse by a single party
- **Representative** SGs ensure a fair share of the priority network among parties
- **Robust** SGs prevent even *partial* shutdown of the priority network by a single party



Inclusive Spanning Graphs

- **Minimal** subgraphs preserving *connectivity*
- May be prioritised for *efficiency* or *security*
- **Diverse** spanning graphs (SGs) prevent *full* network collapse by a single party
- **Representative** SGs ensure a fair share of the priority network among parties
- **Robust** SGs prevent even *partial* shutdown of the priority network by a single party



Minimum Spanning Trees

Newcastle University

Definition

Graph H is an **MST** of G iff:

1. $V_H = V_G$
2. H is connected and acyclic
3. Sum of edge weights is minimum

Kruskal's Algorithm

1. Process edges by increasing weight
2. Add to $E_H \iff$ it doesn't create cycles

Minimum Spanning Trees

Newcastle University

Definition

Graph H is an **MST** of G iff:

1. $V_H = V_G$
2. H is connected and acyclic
3. Sum of edge weights is minimum

Kruskal's Algorithm

1. Process edges by increasing weight
2. Add to $E_H \iff$ it doesn't create cycles

Minimum Spanning Trees

Newcastle University

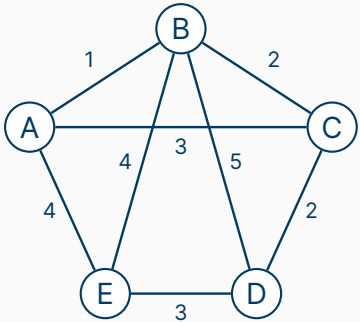
Definition

Graph H is an **MST** of G iff:

1. $V_H = V_G$
2. H is connected and acyclic
3. Sum of edge weights is minimum

Kruskal's Algorithm

1. Process edges by increasing weight
2. Add to $E_H \iff$ it doesn't create cycles



Minimum Spanning Trees

Newcastle University

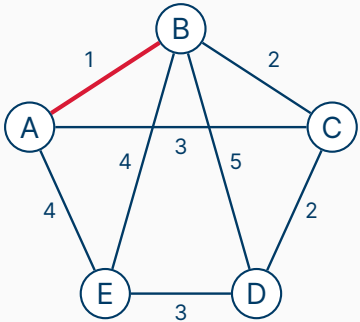
Definition

Graph H is an **MST** of G iff:

1. $V_H = V_G$
2. H is connected and acyclic
3. Sum of edge weights is minimum

Kruskal's Algorithm

1. Process edges by increasing weight
2. Add to $E_H \iff$ it doesn't create cycles



Minimum Spanning Trees

Newcastle University

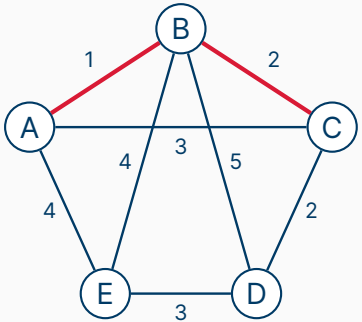
Definition

Graph H is an **MST** of G iff:

1. $V_H = V_G$
2. H is connected and acyclic
3. Sum of edge weights is minimum

Kruskal's Algorithm

1. Process edges by increasing weight
2. Add to $E_H \iff$ it doesn't create cycles



Minimum Spanning Trees

Newcastle University

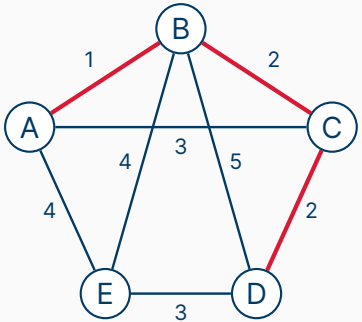
Definition

Graph H is an **MST** of G iff:

1. $V_H = V_G$
2. H is connected and acyclic
3. Sum of edge weights is minimum

Kruskal's Algorithm

1. Process edges by increasing weight
2. Add to $E_H \iff$ it doesn't create cycles



Minimum Spanning Trees

Newcastle University

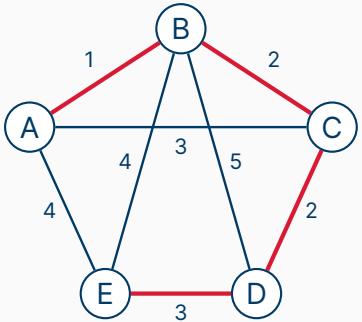
Definition

Graph H is an **MST** of G iff:

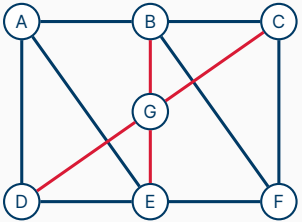
1. $V_H = V_G$
2. H is connected and acyclic
3. Sum of edge weights is minimum

Kruskal's Algorithm

1. Process edges by increasing weight
2. Add to $E_H \iff$ it doesn't create cycles



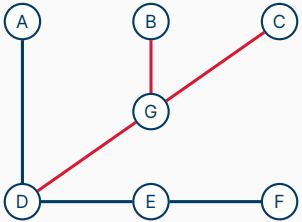
Diversity vs Representativeness



Original Graph

8 Blue, 4 Red

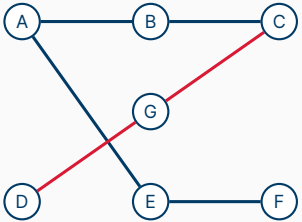
Ratio 2 : 1



Diverse MST

3 Blue, 3 Red

Ratio 1 : 1



Representative MST

4 Blue, 2 Red

Ratio 2 : 1

Diverse MST Algorithm

Newcastle University

1: Initialise graph $T = (V, E_T = \emptyset)$
2: **while** $|E_T| < |V| - 1$ **do**
3: **for** $e \in E' = E \setminus E_T$ **do**
4: $S(e) = (1-d) \cdot \frac{w(e)}{\max\limits_{\epsilon \in E_G} w(\epsilon)} + d \cdot \frac{|E_T^{c(e)}|}{|E_T|}$
5: **for** $e \in \texttt{sorted}(E')$ **do**
6: **if** $E_T + \{e\}$ is acyclic **then**
7: $E_T \leftarrow E_T + \{e\}$
8: **break**
9: **return** T

Representative MST Algorithm

Newcastle University

1: Initialise graph $T = (V, E_T = \emptyset)$
2: **while** $|E_T| < |V| - 1$ **do**
3: **for** $e \in E' = E \setminus E_T$ **do**
4: $S(e) = (1-d) \cdot \frac{w(e)}{\max\limits_{\epsilon \in E_G} w(\epsilon)} + d \cdot \frac{\varphi_{c(e)}}{\max\limits_{c \in C} \varphi_c}$ where $\varphi_c = \frac{p_T(c)}{p_G(c)}$
5: **for** $e \in \texttt{sorted}(E')$ **do**
6: **if** $E_T + \{e\}$ is acyclic **then**
7: $E_T \leftarrow E_T + \{e\}$
8: **break**
9: **return** T

The Effect of Trade-Off Parameter d

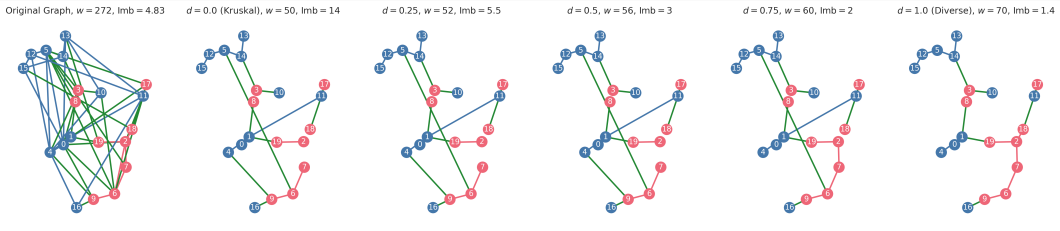
$$S(e) = (1-d) \cdot \frac{w(e)}{\max\limits_{\epsilon \in E_G} w(\epsilon)} + d \cdot \frac{|E_T^{c(e)}|}{|E_T|}$$

The Effect of Trade-Off Parameter d

$$S(e) = (1-d) \cdot \frac{w(e)}{\max\limits_{\epsilon \in E_G} w(\epsilon)} + d \cdot \frac{|E_T^{c(e)}|}{|E_T|}$$

The Effect of Trade-Off Parameter d

$$S(e) = (1-d) \cdot \frac{w(e)}{\max\limits_{\epsilon \in E_G} w(\epsilon)} + d \cdot \frac{|E_T^{c(e)}|}{|E_T|}$$



Relevant Metrics

Newcastle University

Efficiency – Weight Sum

$$W(G) = \sum_{e \in E} w(e)$$

Lower $\rightarrow$ More Efficient

Diversity – Normalised Entropy

$$NE(G) = \frac{-\sum_{c \in C} p_G(c) \log p_G(c)}{\log |C|}$$

Low 0 $\rightarrow$ 1 High Diversity

Representativeness – Proportion Similarity

$$Sim(G, H) = 1 - \sum_{c \in C} |p_G(c) - p_H(c)|$$

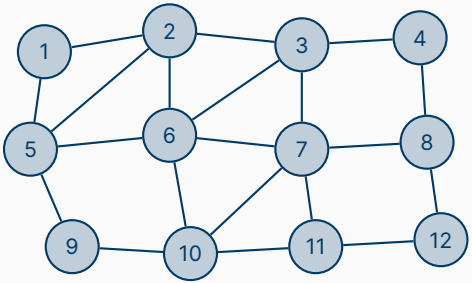
Less 0 $\rightarrow$ 1 More Similar

Synthetic Graph Generators

Newcastle University

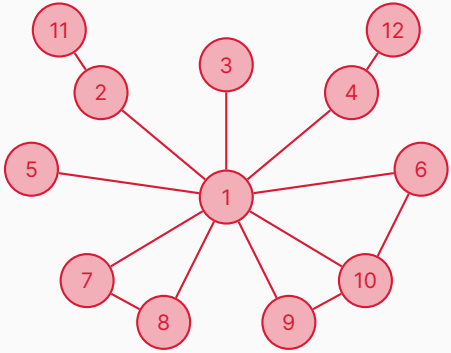
Erdös–Rényi

Edges are present independently with probability p

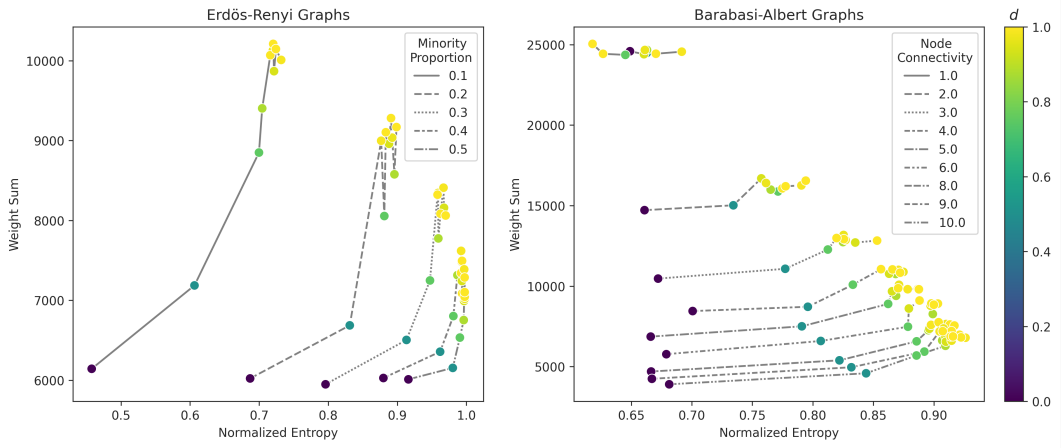


Barábasi-Albert

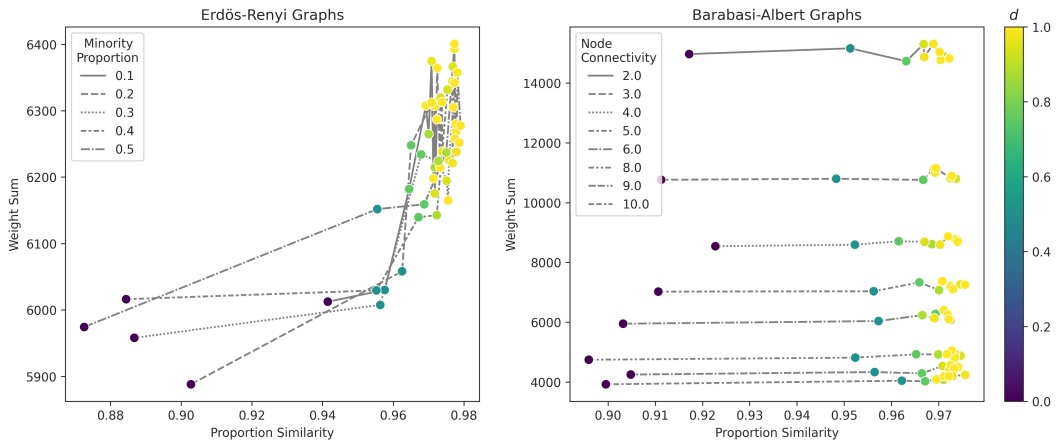
New nodes attach preferentially to already popular nodes



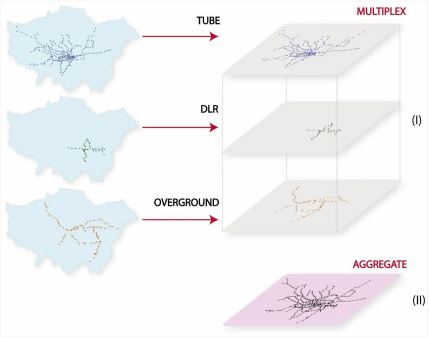
Diversity / Efficiency Trade-Off



Representativeness / Efficiency Trade-Off



London Transport Network Data



Algorithm	Weight Sum	Prop. Similarity	Norm. Entropy
Kruskal (MST)	**429890**	0.959	0.762
Representative	431422	**0.975**	0.751
Diverse	436429	0.926	**0.781**

Work in Progress

Newcastle University

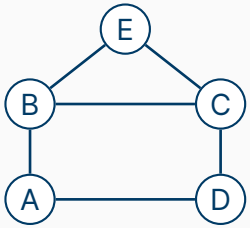
Definition

$H \subseteq G$ is a ***t*-spanner** of G iff, for every pair of vertices $u, v \in V$,

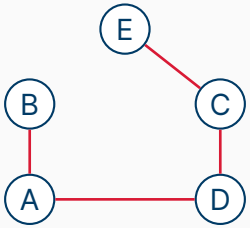
$$d_H(u, v) \leqslant t \cdot d_G(u, v),$$

where $t \geqslant 1$ is the **stretch factor**.

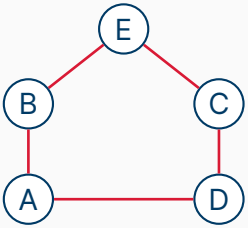
Original Graph



Min Spanning Tree



Min 2-Spanner



The Greedy t-Spanner Algorithm

Process E_G by increasing weight

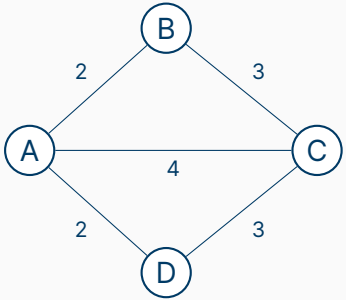
1. Initialise $H = (V_G, \varnothing)$
2. For each uv in E_G:

   if $d_H(u, v) > t \cdot w(uv) \Rightarrow$ add uv to E_H.

Adding Inclusive Constraints

Sort by diverse or representative score, as with MSTs.

Original Graph



The Greedy t-Spanner Algorithm

Process E_G by increasing weight

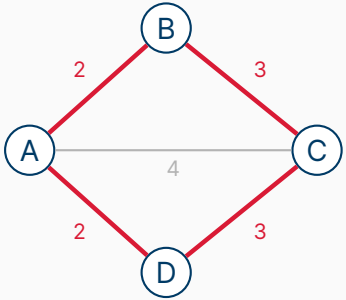
1. Initialise $H = (V_G, \varnothing)$
2. For each uv in E_G:

   if $d_H(u, v) > t \cdot w(uv) \Rightarrow$ add uv to E_H.

Adding Inclusive Constraints

Sort by diverse or representative score, as with MSTs.

Greedy 2-Spanner



The Greedy t-Spanner Algorithm

Process E_G by increasing weight

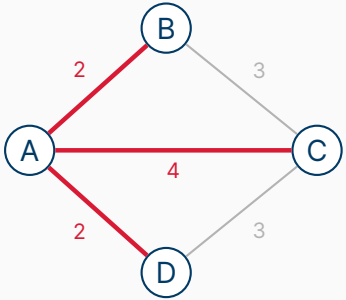
1. Initialise $H = (V_G, \varnothing)$
2. For each uv in E_G:

   if $d_H(u, v) > t \cdot w(uv) \Rightarrow$ add uv to E_H.

Adding Inclusive Constraints

Sort by diverse or representative score, as with MSTs.

Minimum 2-Spanner



The Greedy t-Spanner Algorithm

Process E_G by increasing weight

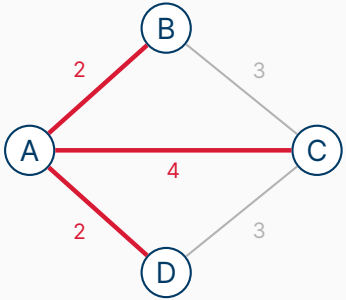
1. Initialise $H = (V_G, \varnothing)$
2. For each uv in E_G:

$$\text{if } d_H(u, v) > t \cdot w(uv) \Rightarrow \text{add } uv \text{ to } E_H.$$

Adding Inclusive Constraints

Sort by diverse or representative score, as with MSTs.

Minimum 2-Spanner



Colour-Robust Graphs

Newcastle University

Motivation

- A train network is operated by n independent companies
- Each company has certain allocated routes
- **Goal:** If a company stops working, the network **stays connected**

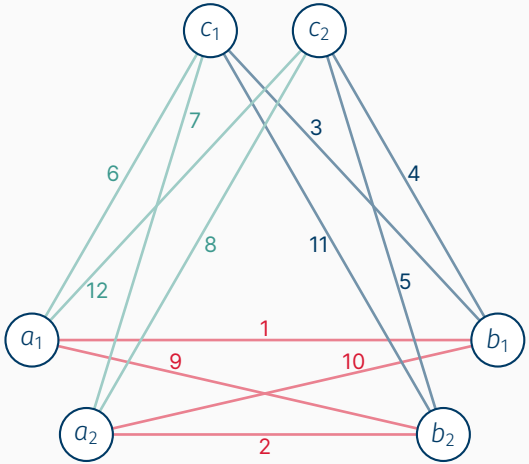
Definition

An edge-coloured graph $G = (V, E)$ is **colour robust** if $G' = (V, E \setminus E_c)$ is connected for every colour $c \in C$.

Greedy Colour-Robust SG Algorithm

Process edges in increasing weight.
For each edge uv of colour i:

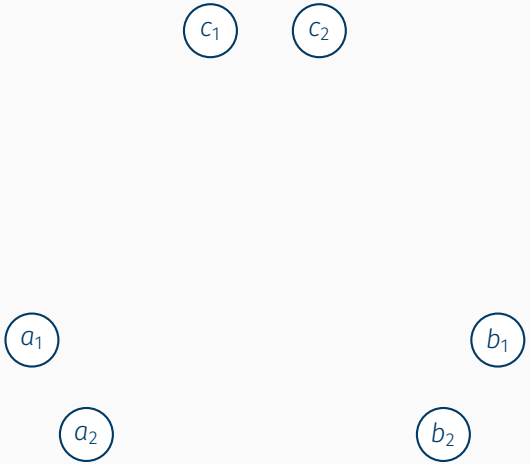
1. For each other colour $j \neq i$, remove all j-edges from the current graph.
2. If (u, v) are disconnected for at least one such colour removal, add uv; otherwise, skip it.



Greedy Colour-Robust SG Algorithm

Process edges in increasing weight.
For each edge uv of colour i:

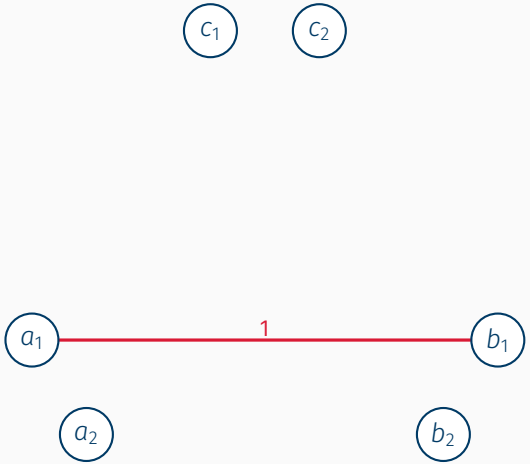
1. For each other colour $j \neq i$, remove all j-edges from the current graph.
2. If (u, v) are disconnected for at least one such colour removal, add uv; otherwise, skip it.



Greedy Colour-Robust SG Algorithm

Process edges in increasing weight.
For each edge uv of colour i:

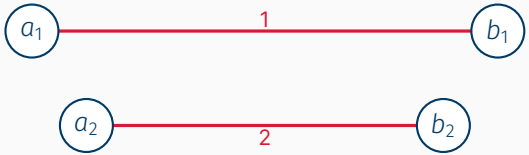
1. For each other colour $j \neq i$, remove all j-edges from the current graph.
2. If (u, v) are disconnected for at least one such colour removal, add uv; otherwise, skip it.



Greedy Colour-Robust SG Algorithm

Process edges in increasing weight.
For each edge uv of colour i:

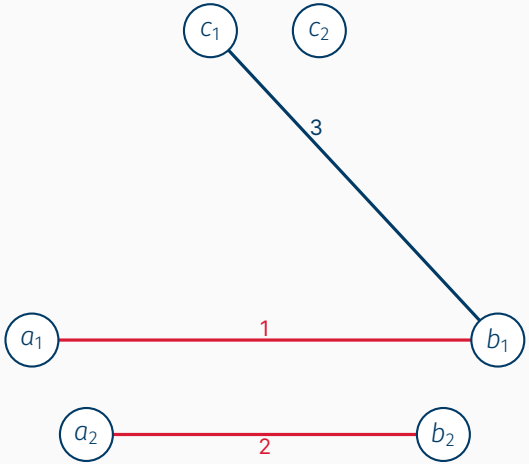
1. For each other colour $j \neq i$, remove all j-edges from the current graph.
2. If (u, v) are disconnected for at least one such colour removal, add uv; otherwise, skip it.



Greedy Colour-Robust SG Algorithm

Process edges in increasing weight.
For each edge uv of colour i:

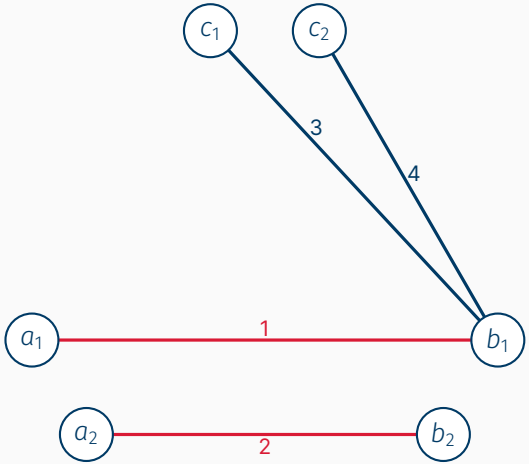
1. For each other colour $j \neq i$, remove all j-edges from the current graph.
2. If (u, v) are disconnected for at least one such colour removal, add uv; otherwise, skip it.



Greedy Colour-Robust SG Algorithm

Process edges in increasing weight.
For each edge uv of colour i:

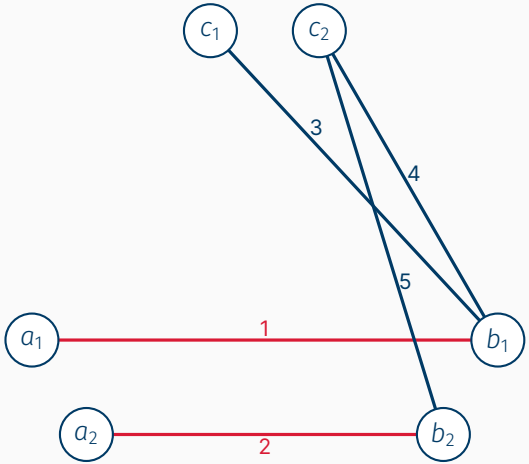
1. For each other colour $j \neq i$, remove all j-edges from the current graph.
2. If (u, v) are disconnected for at least one such colour removal, add uv; otherwise, skip it.



Greedy Colour-Robust SG Algorithm

Process edges in increasing weight.
For each edge uv of colour i:

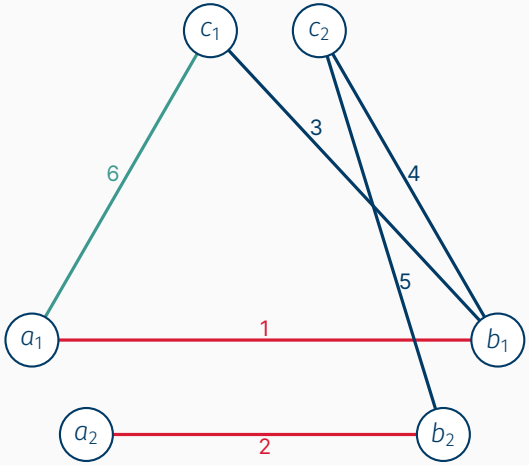
1. For each other colour $j \neq i$, remove all j-edges from the current graph.
2. If (u, v) are disconnected for at least one such colour removal, add uv; otherwise, skip it.



Greedy Colour-Robust SG Algorithm

Process edges in increasing weight.
For each edge uv of colour i:

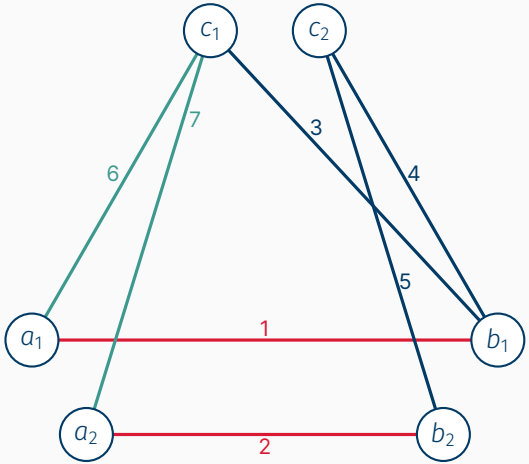
1. For each other colour $j \neq i$, remove all j-edges from the current graph.
2. If (u, v) are disconnected for at least one such colour removal, add uv; otherwise, skip it.



Greedy Colour-Robust SG Algorithm

Process edges in increasing weight.
For each edge uv of colour i:

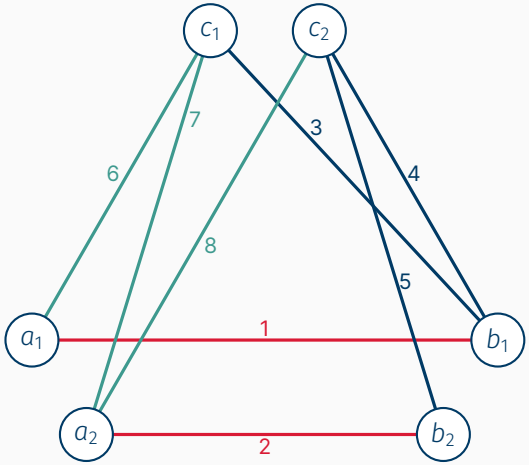
1. For each other colour $j \neq i$, remove all j-edges from the current graph.
2. If (u, v) are disconnected for at least one such colour removal, add uv; otherwise, skip it.



Greedy Colour-Robust SG Algorithm

Process edges in increasing weight.
For each edge uv of colour i:

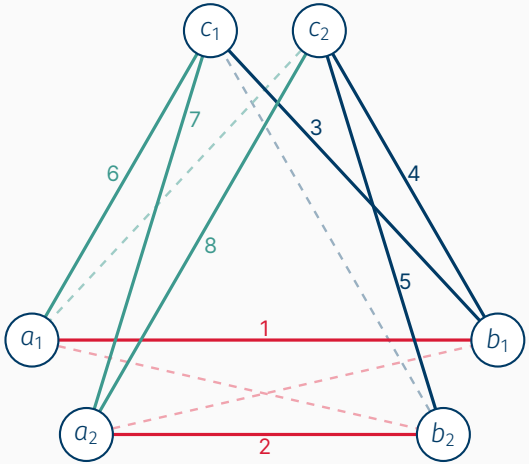
1. For each other colour $j \neq i$, remove all j-edges from the current graph.
2. If (u, v) are disconnected for at least one such colour removal, add uv; otherwise, skip it.



Greedy Colour-Robust SG Algorithm

Process edges in increasing weight.
For each edge uv of colour i:

1. For each other colour $j \neq i$, remove all j-edges from the current graph.
2. If (u, v) are disconnected for at least one such colour removal, add uv; otherwise, skip it.



Where We Are

- Ongoing collaboration with D. Archambault, M. Chimani, I. Dixon, G. Liotta, M. Nöllenburg, T. Piselli, J. Salas, A. Tappini, and M. Wallinger
- Implemented both t-spanners and colour-robust spanners
- Perform experiments on airline alliance flights dataset
- Working on theoretical results
- Extend to other relevant algorithms