

Analyzing Large Graphs with `QuasiStableColors.jl`



Accelerating Max-flow, Linear Programs and Centrality

Moe Kayali, Dan Suciu

University of Washington, Seattle

JuliaCon '23 — July 27, 2023

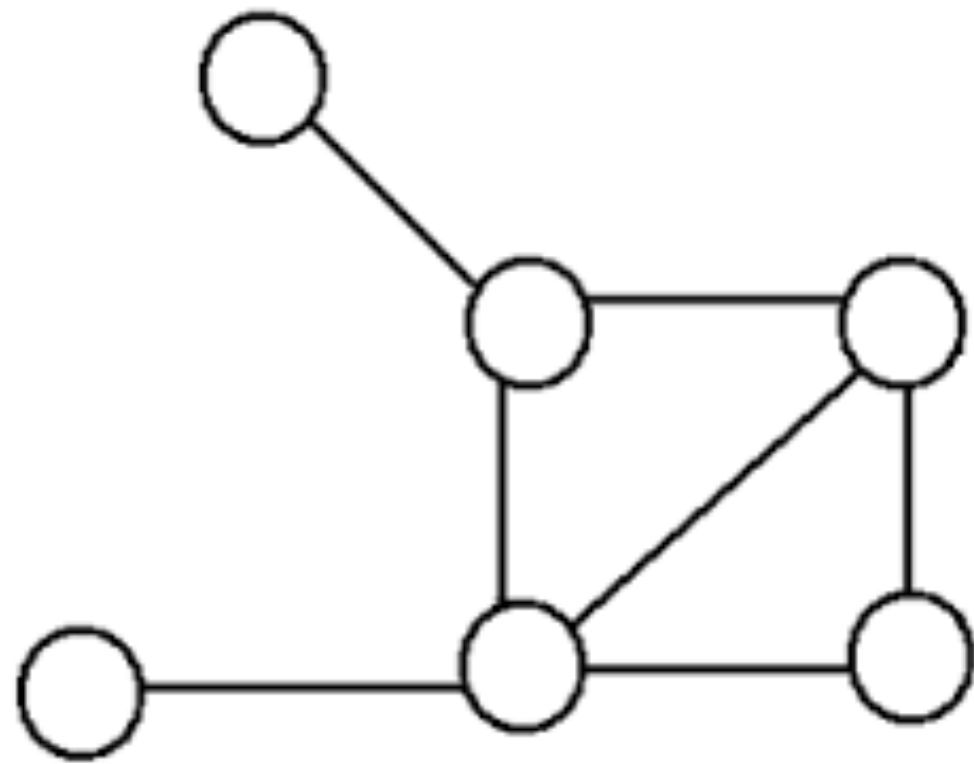


Motivation

Graphs are:

Motivation

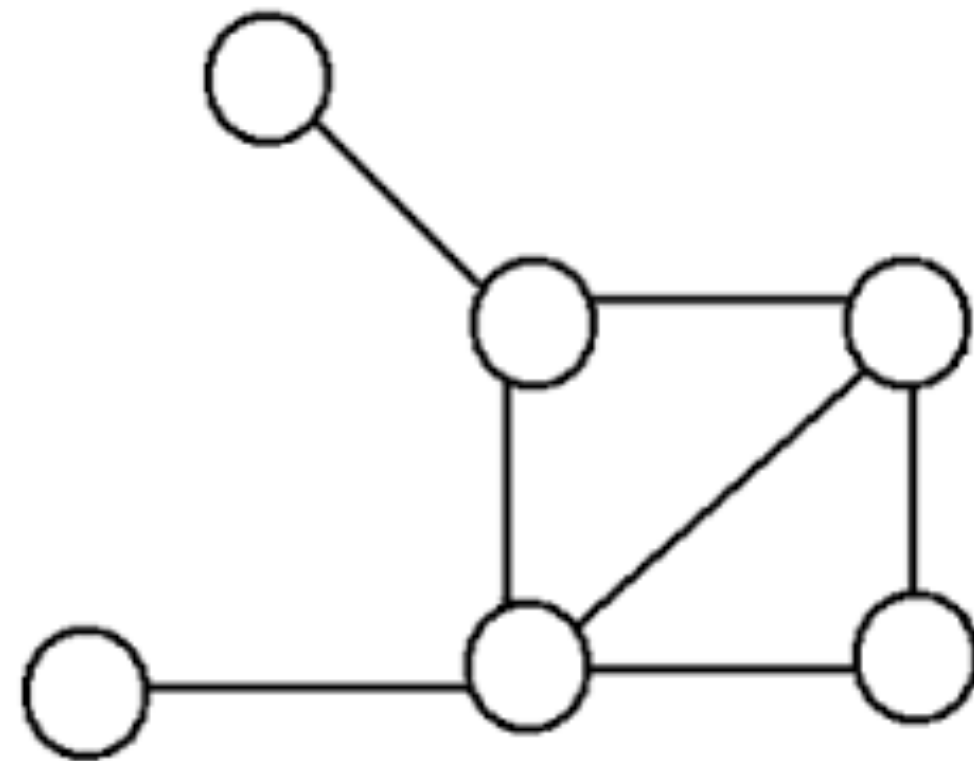
Graphs are:



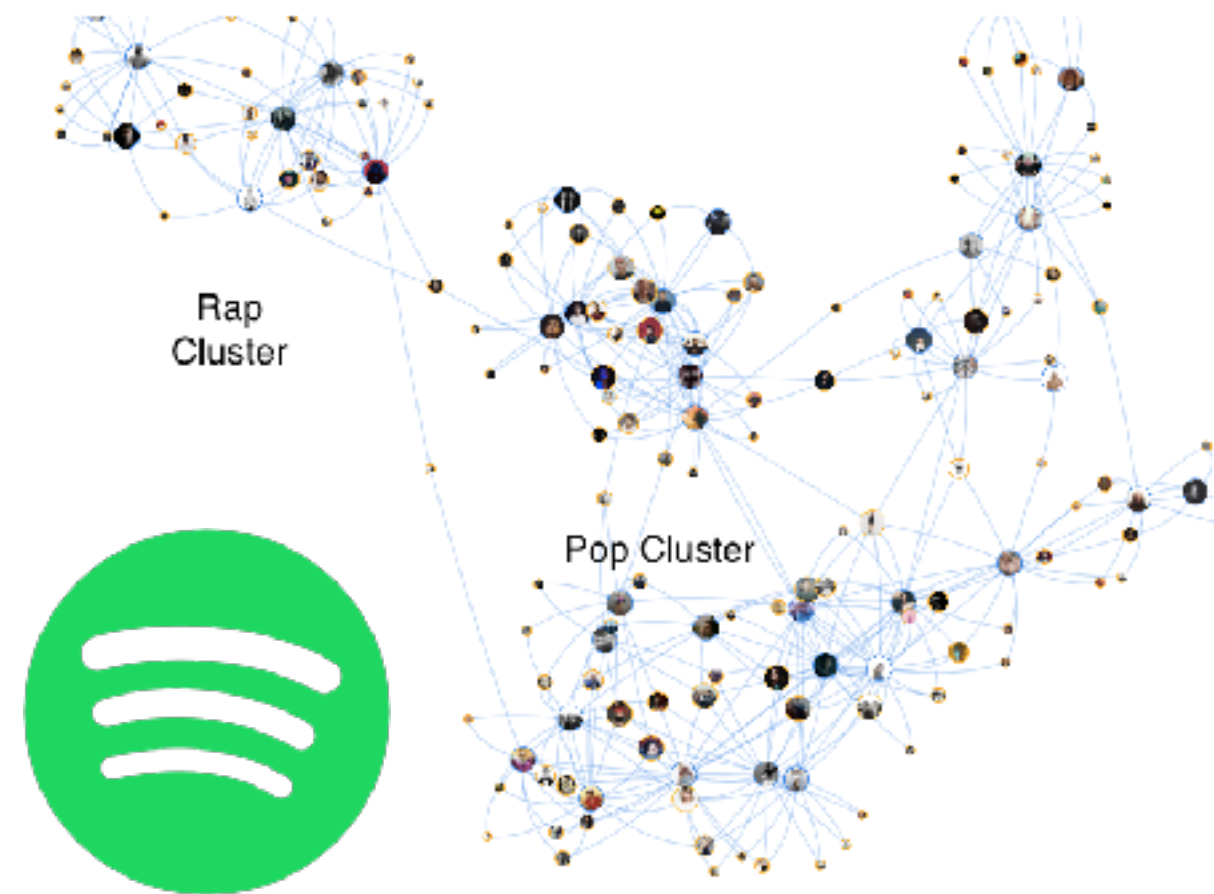
**A fundamental CS
building block**

Motivation

Graphs are:



**A fundamental CS
building block**

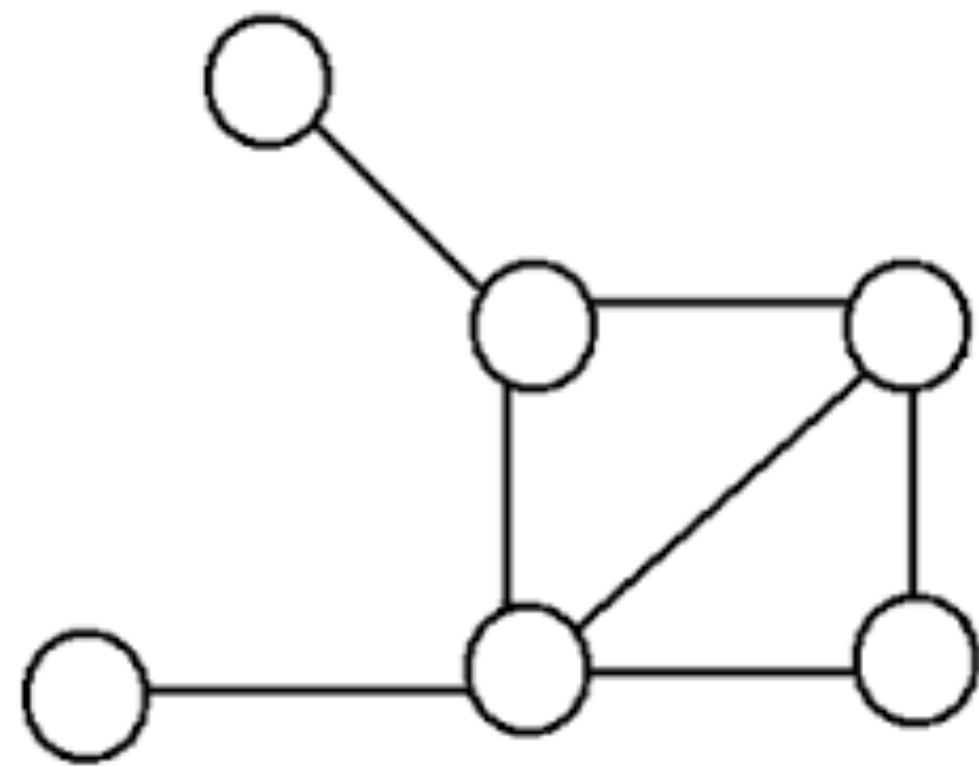


Spotify GNN-based Recommender

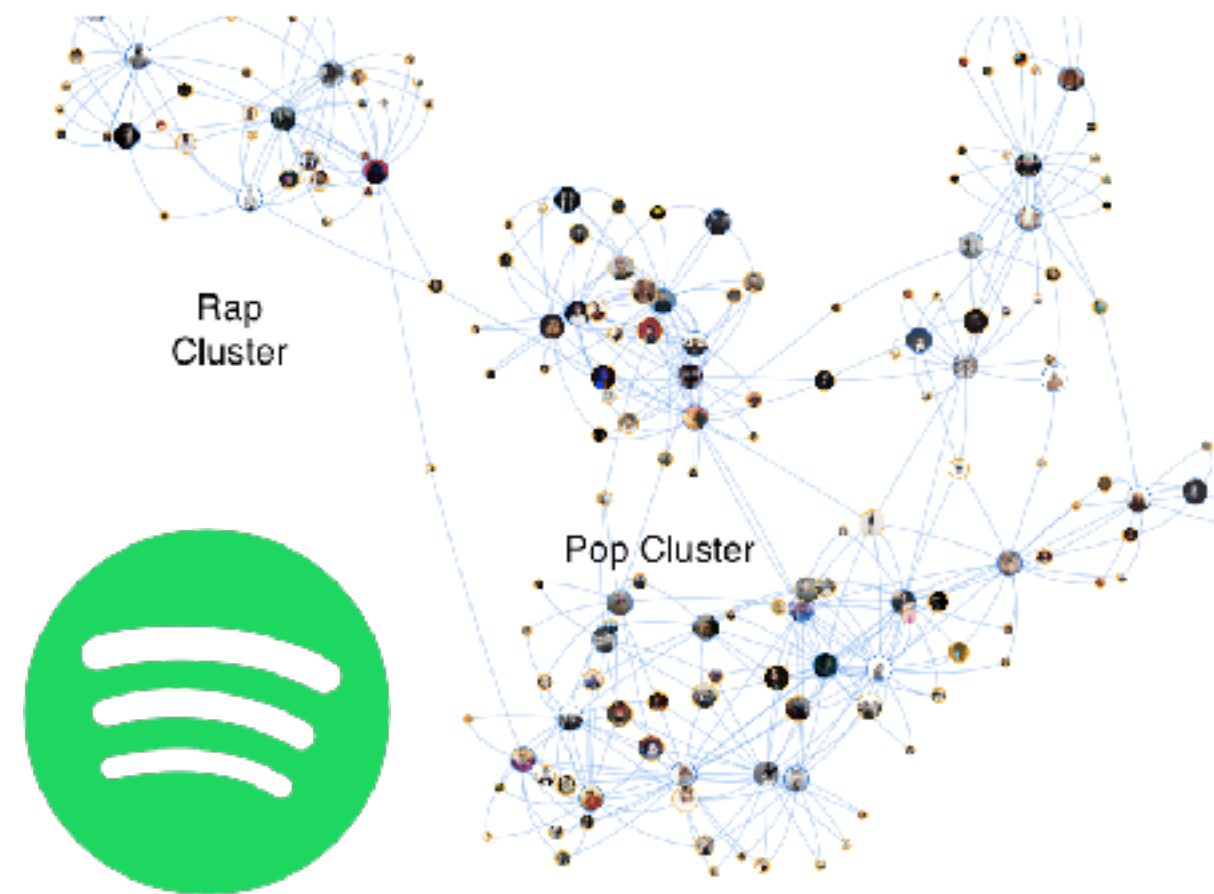
**Increasingly
applicable**

Motivation

Graphs are:

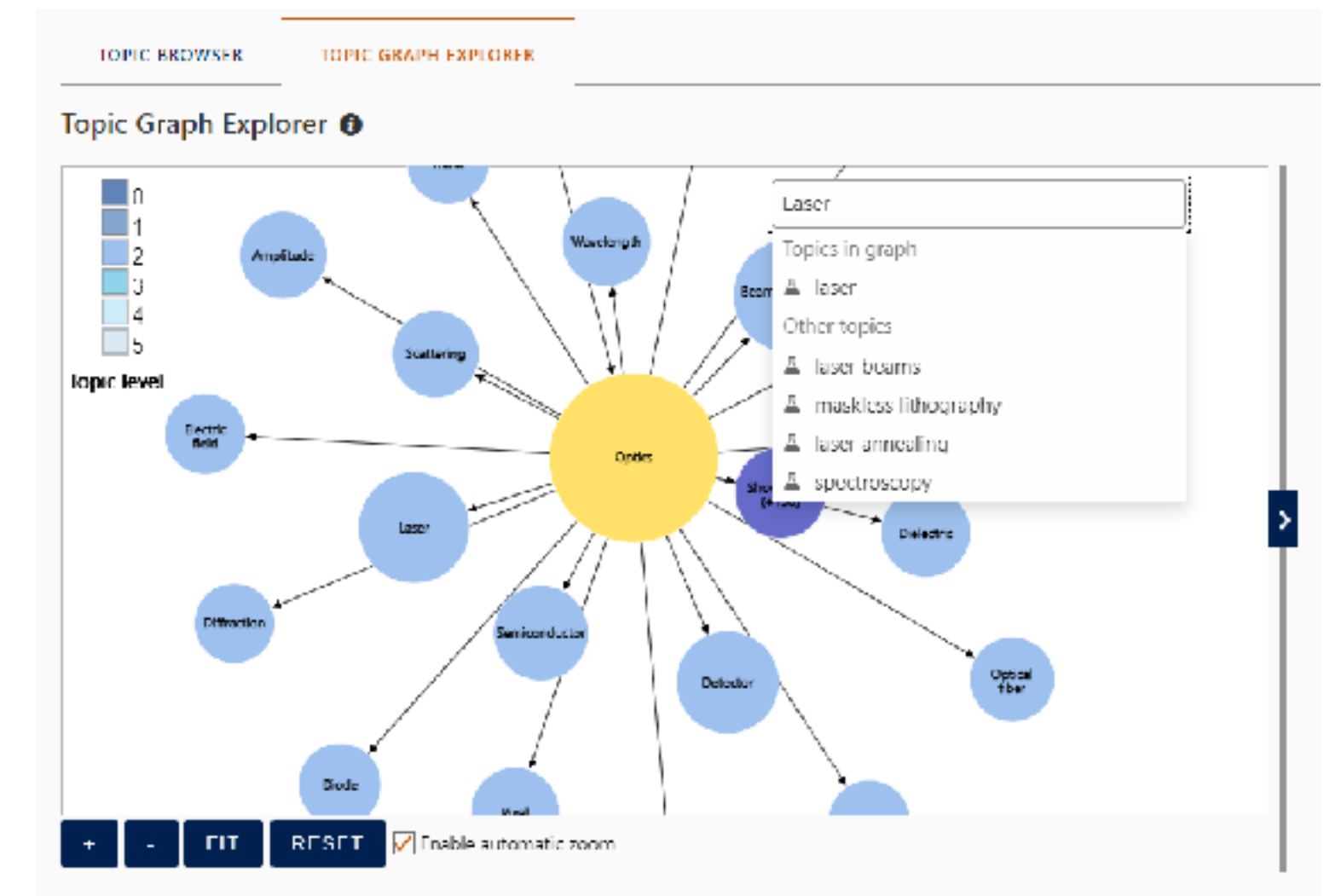


**A fundamental CS
building block**



Spotify GNN-based Recommender

**Increasingly
applicable**



Microsoft Academic Graph, >1B nodes

**Increasingly large,
complex**

Motivation

Developed measures on graphs:

Motivation

Developed measures on graphs:

Maximum-flow/minimum-cut



Traffic Modeling

Motivation

Developed measures on graphs:

Maximum-flow/minimum-cut



Traffic Modeling

Centrality



Landmark, Fraud Detection

However...

Despite the increasingly larger graphs practitioners encounter, **scalability of graph analysis methods remains poor.**

Betweenness Centrality Performance

Node count	Runtime
100	5 milliseconds
1,000	270 milliseconds
10,000	4 minutes
100,000	42 minutes

The Library

The Library



QuasiStableColors.jl

The Library

Implementation of a novel graph approximation approach, achieving average 10-100x speedup with less than <15% error



QuasiStableColors.jl

The Library

Implementation of a novel graph approximation approach, achieving average 10-100x speedup with less than <15% error



QuasiStableColors.jl

*“Quasi-stable Coloring for Graph Compression”
appearing in Very Large Databases Conference (VLDB) 2023*

ABSTRACT

We propose *quasi-stable coloring*, an approximate version of stable coloring. Stable coloring, also called color refinement, is a well-studied technique in graph theory for classifying vertices, which can be used to build compact, lossless representations of graphs. However, its usefulness is limited due to its reliance on strict symmetries. Real data compresses very poorly using color refinement. We propose the first, to our knowledge, approximate color refinement scheme, which we call quasi-stable coloring, and allow for a tradeoff between the degree of compression and the accuracy of the representation. We study three applications: Linear Programming, Max-Flow, and Betweenness Centrality, and provide theoretical evidence in each case that a quasi-stable coloring can lead to good approximations on the reduced graph. Next, we consider how to compute a maximal quasi-stable coloring: we prove that, in general, this problem is NP-hard, and propose a simple, yet effective algorithm based on heuristics. Finally, we evaluate experimentally the quasi-stable coloring technique on several real graphs and applications, comparing with prior approximation techniques.

PVLDB Reference Format:
Moe Kayali and Dan Suciu. Quasi-stable Coloring for Graph Compression. PVLDB, 16(4): 803 - 815, 2022.
doi:10.14778/3574245.3574264

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at <https://github.com/mkyl/QuasiStableColors.jl>.

Quasi-stable Coloring for Graph Compression

Approximating Max-Flow, Linear Programs, and Centrality

Moe Kayali
University of Washington
kayali@cs.washington.edu

Dan Suciu
University of Washington
suciu@cs.washington.edu

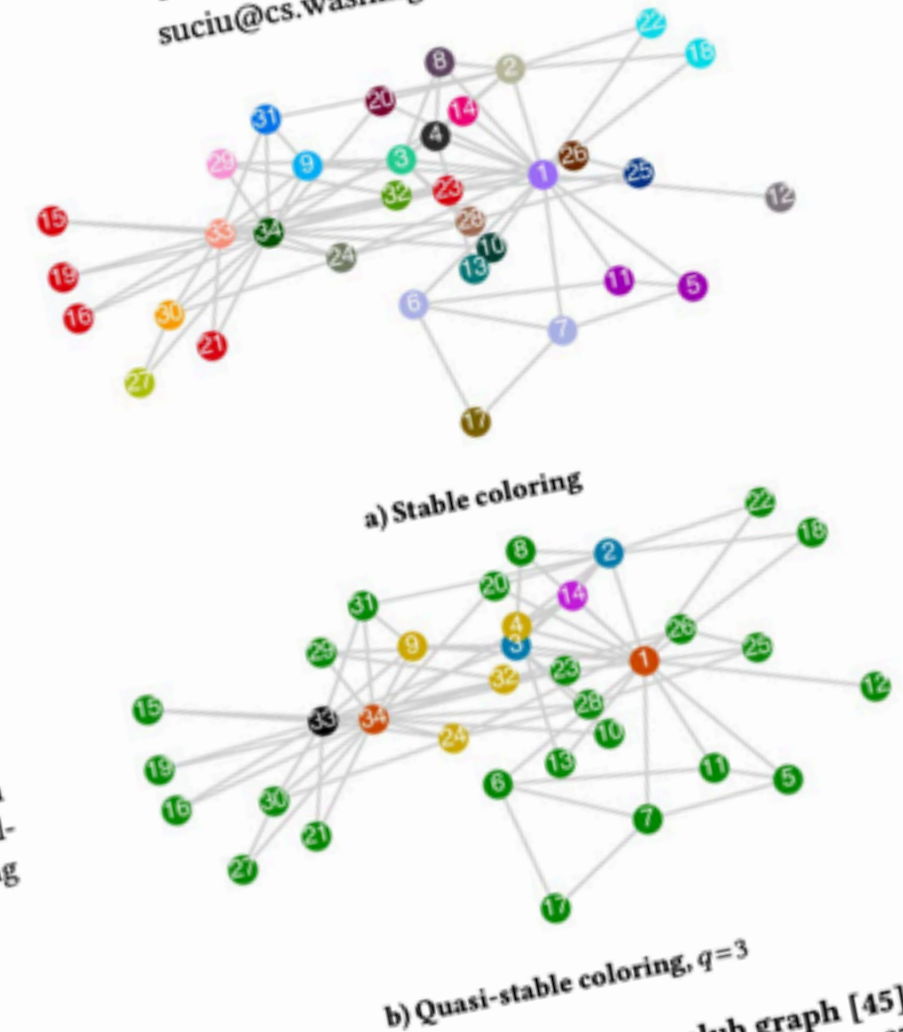


Figure 1: Coloring Zachary's karate club graph [45] where $|V|=34$, $|E|=78$. While the stable coloring requires 27 colors, for a quasi-stable color 6 colors suffice when $q=3$. Note in 1b the club leaders {1,34} are put into their own color.

computed efficiently, in almost linear time [34], can be generalized to labeled graphs, weighted graphs, directed or undirected graphs, and multigraphs, and is used by graph isomorphism algorithms [17], in Networks [12, 29, 31, 44]. We review its refinement and its

API

[QuasiStableColors.Centrality.approx_betweenness_centrality](#) — Function

```
approx_betweenness_centrality(  
    G::Graph,  
    q::Number,  
    n_colors::Int,  
)
```

Approximate betweenness centrality using a q-stable coloring with maximum error `q` or size `n_colors`, whichever is smaller.

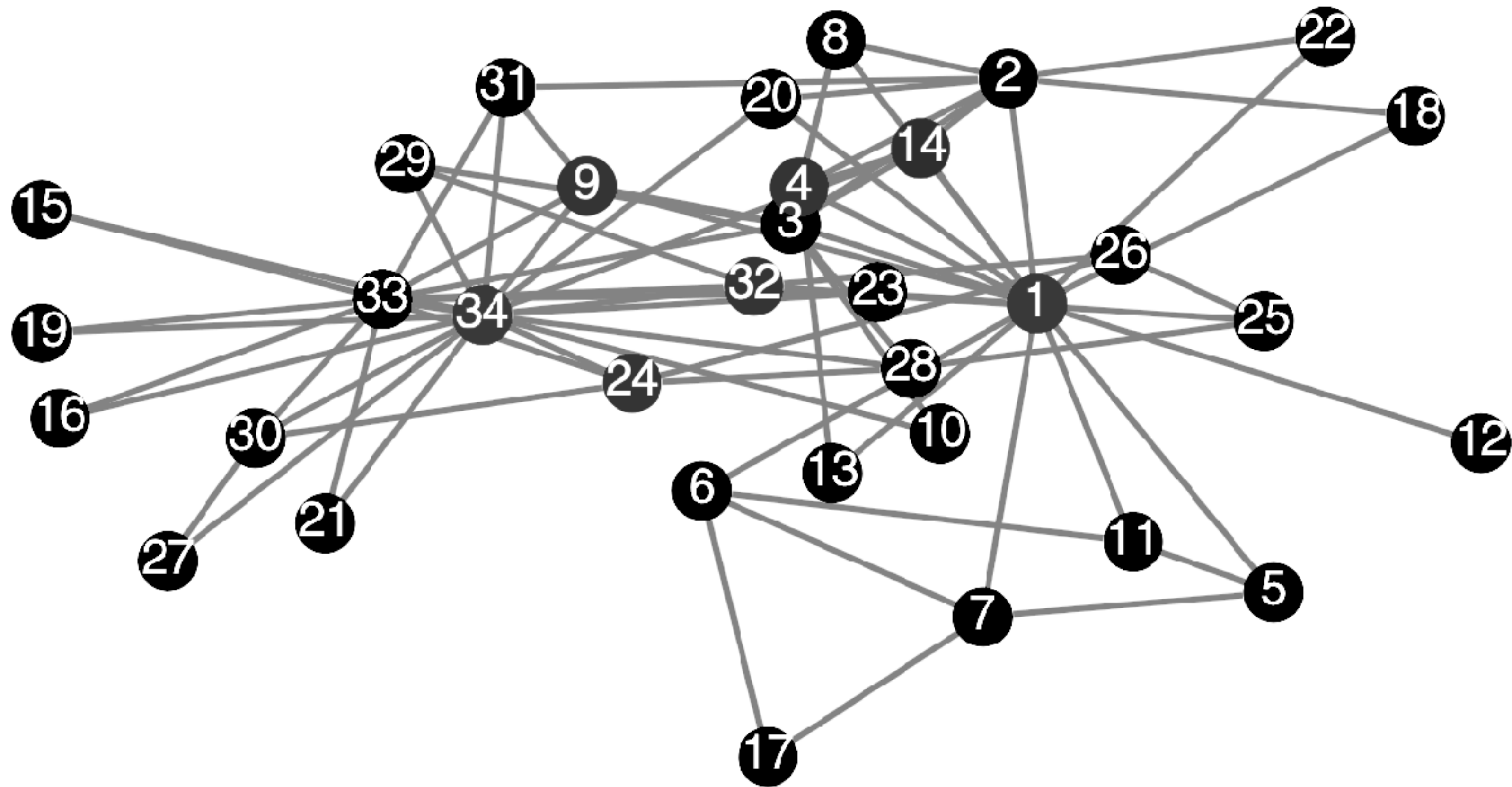
[QuasiStableColors.Optimize.lifted_minimize](#) — Function

```
lifted_minimize(  
    A,  
    b::Vector,  
    c::Vector,  
    q=0.0,  
    n_colors=Inf,  
)
```

Approximate the linear program $\min c^T x$ where $Ax \geq b, x \geq 0$.

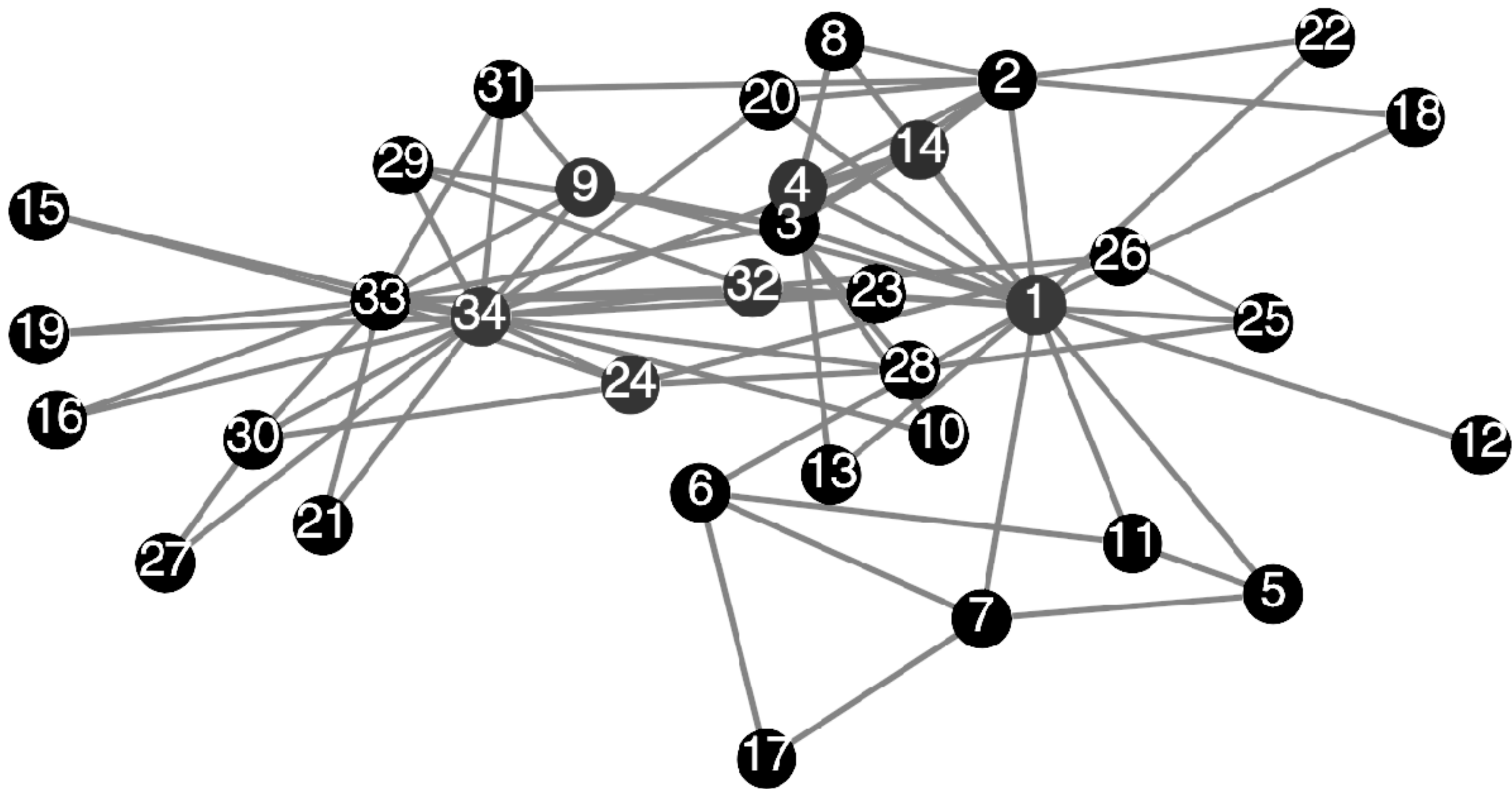
Uses a quasi-stable coloring with maximum error `q` or `n_colors` colors, whichever is smaller.

[QuasiStableColors.Flow.lifted_maxflow](#) — Function



Karate Club

W. W. Zachary. An information flow model for conflict and fission in small groups. Journal of anthropological research, 33(4):452–473, 1977.



Karate Club

$$|V| = 34, |E| = 78$$

QuasiStableColors.q_color — Method

```
q_color(  
  G::AbstractGraph{T},  
  q = 0.0,  
  n_colors = Inf,  
  weights::SparseMatrixCSC{<:Number,Int} = nothing,  
  special = Set{T}(),  
  warm_start = Vector{Vector{T}}(),  
)
```

Compute a quasi-stable coloring for the graph `G`. Typically, you should set one of:

- `q`: maximum q-error allowed
- `n_colors`: number of colors to use

Advanced, optional parameters:

- `weights`: edge weights to use
- `weighting`: whether to prioritize larger colors (i.e. with more members).

`false` to prioritize colors with the largest error, regardless of color size.

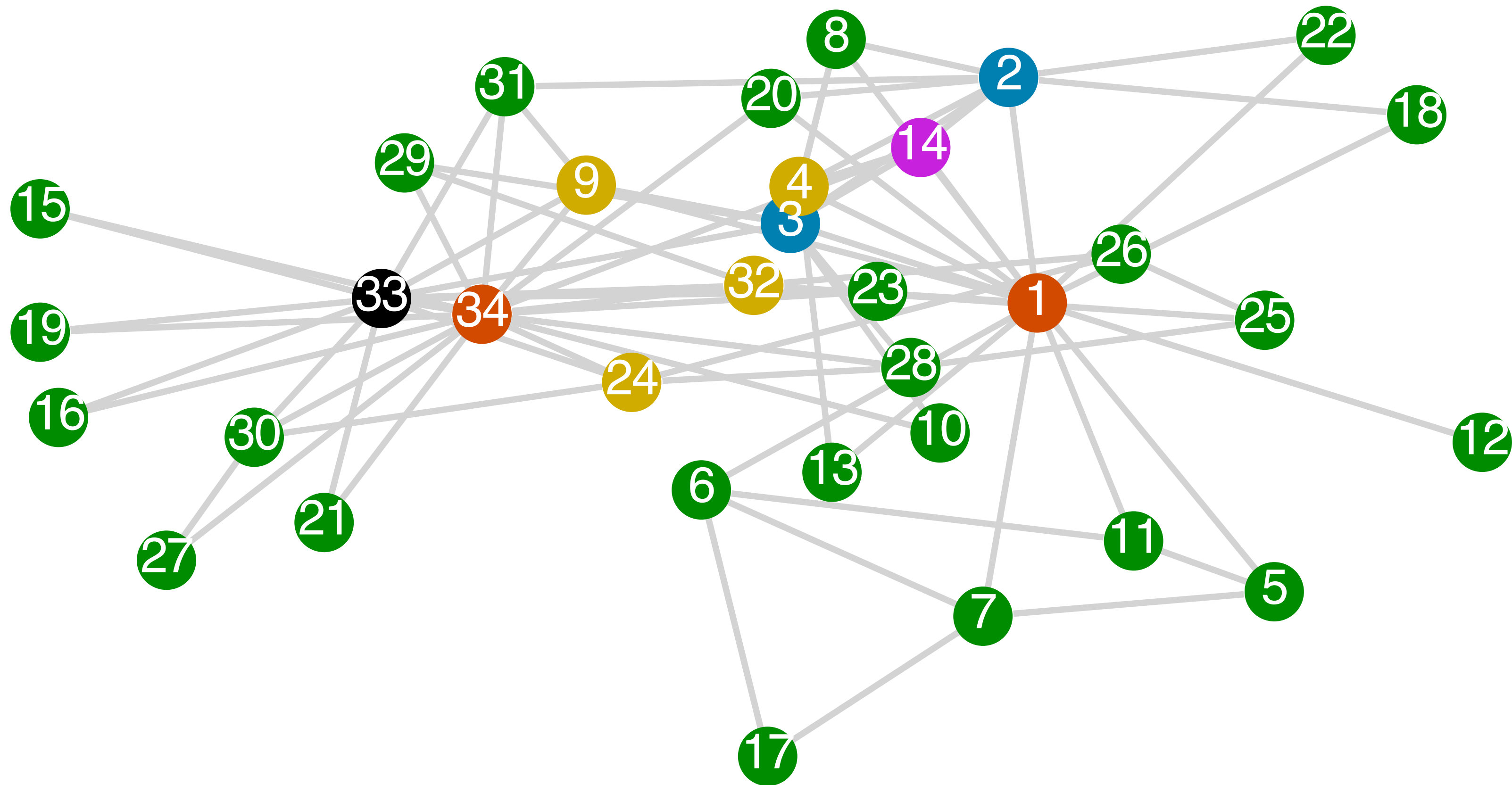
- `warm_start`: coloring to refine. If not provided, starts using coarsest

(single color) partitioning.

- `special`: node IDs which will always get the

```
C = QSC.q_color(g, q=1.0)
```

```
Quasi-stable coloring(color count=4, max q-error=1.0)
```



Karate Club

$|V| = 34, |E| = 78, q = 3.0, |P| = 6$

Quasi-Stable Coloring

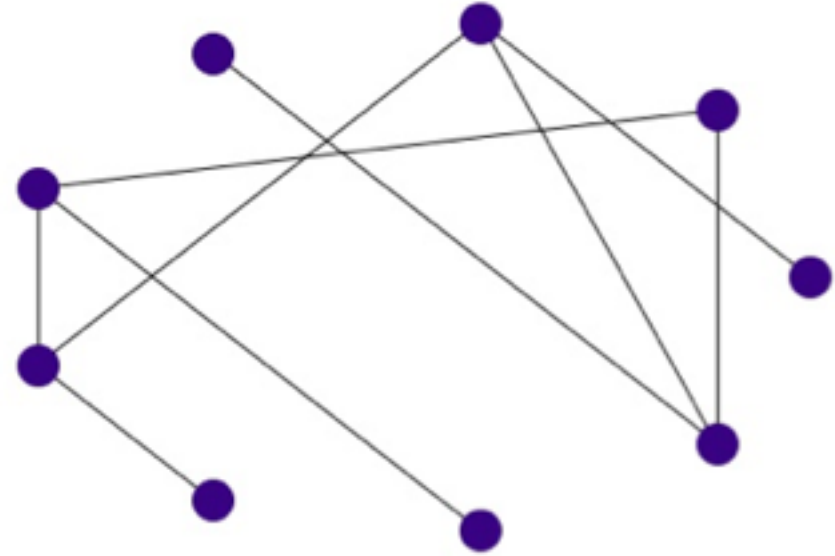
A generalization of stable coloring.

DEFINITION. (*Quasi-stable Coloring*) Call a coloring $P = \{P_1, \dots, P_k\}$ quasi-stable if, for some $q \geq 0$:

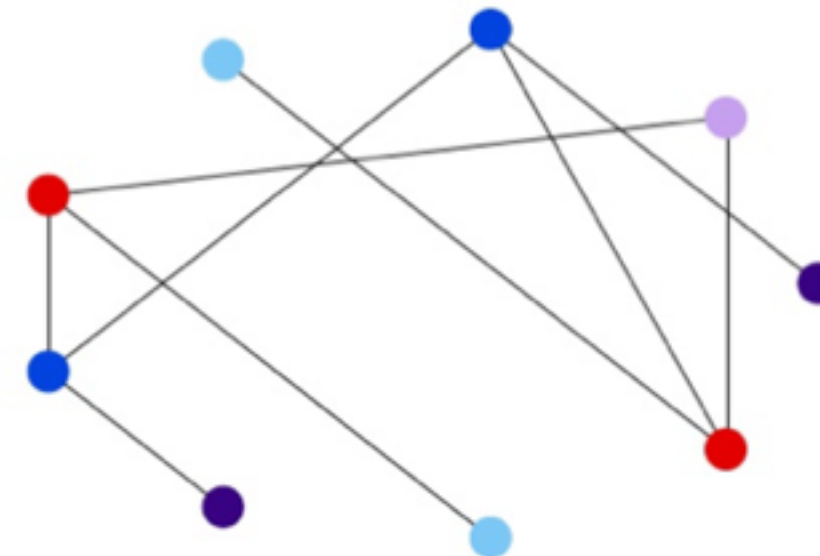
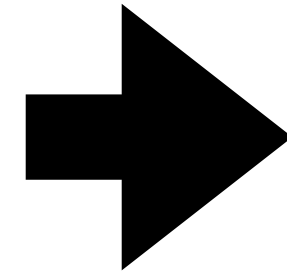
$$\forall i, j, \forall x, y \in P_i : -q \leq |N(x) \cap P_j| - |N(y) \cap P_j| \leq q$$

where $N(v)$ denotes the neighbors of vertex v .

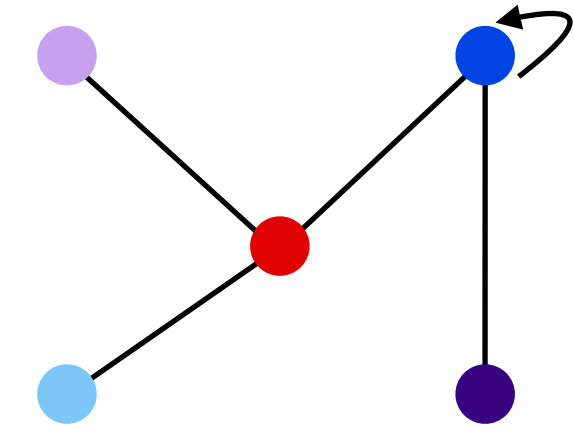
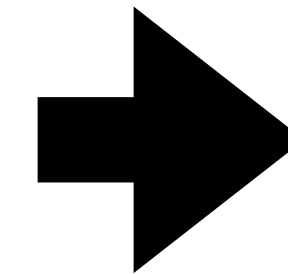
Compression



Initial Graph

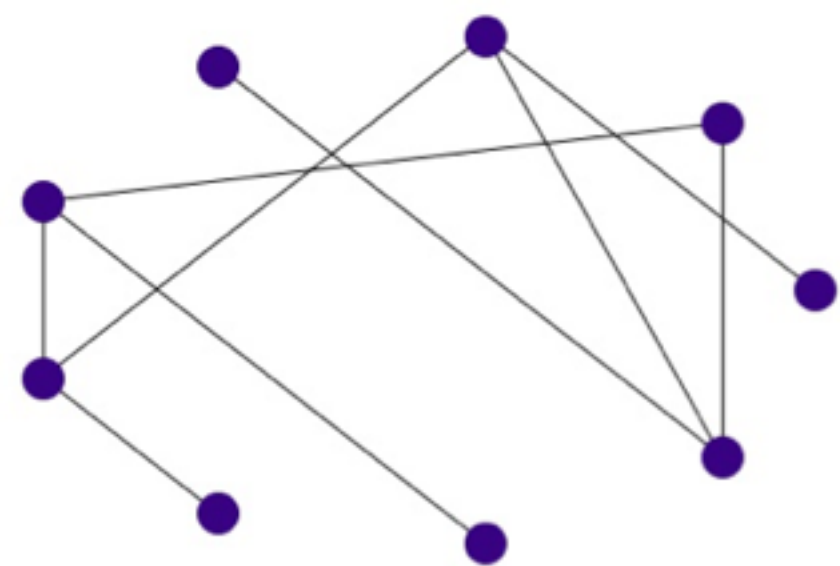


Quasi-Stable Coloring

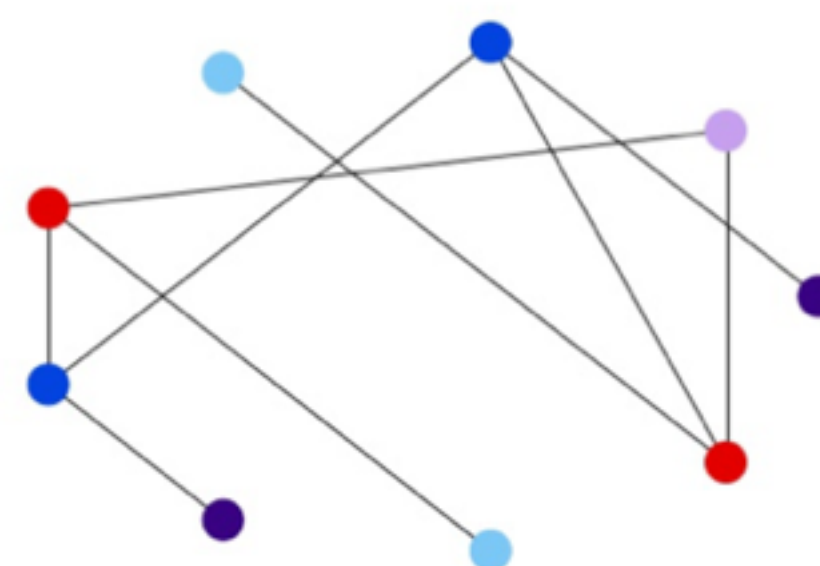
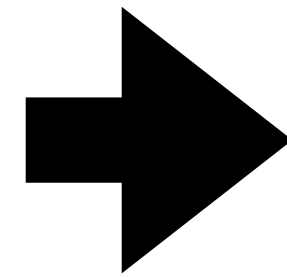


**Lifted Graph,
Quotient Graph**

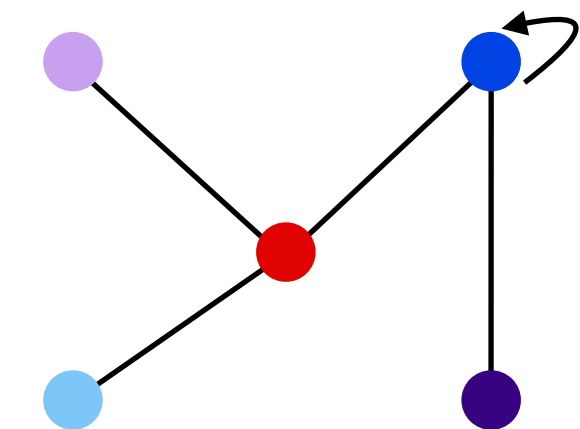
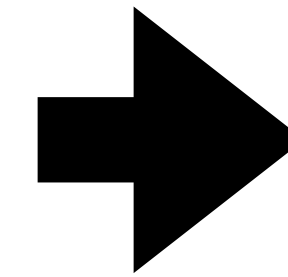
Compression



Initial Graph



Quasi-Stable Coloring



**Lifted Graph,
Quotient Graph**

*Adapted from
M. Grohe. word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data. PODS 2020, Portland, OR, USA, June 14-19, 2020, pages 1–16. ACM, 2020.*

Applications

(1) Max-flow/Min-cut

(2) Linear Optimization

(3) Betweenness Centrality



The Catch

Theorem: Computing a *maximal* quasi-stable coloring is NP-complete, and similarly for any ε -stable coloring.

Experiments

Datasets and Setup

Table 2: Summary of graphs used for evaluation

Name	Vertices	Edges	Real/ Sim.	Source
<i>General evaluation</i>				
Karate	34	75	R	[10]
OpenFlights	3 425	38 513	R	[35]
DBLP	317 080	1 049 866	R	[7]
<i>Centrality</i>				
Astrophysics	18 772	198 110	R	[23]
Facebook	22 470	171 002	R	[26]
Deezer	28 281	92 752	R	[38]
Enron	36 692	183 831	R	[21]
Epinions	75 879	508 837	R	[36]
<i>Maximum-flow</i>				
Tsukuba0	110 594	506 546	R	[32]
Tsukuba2	110 594	500 544	R	[32]
Venus0	166 224	787 946	R	[39]
Venus1	166 224	787 716	R	[39]
Sawtooth0	164 922	790 296	R	[39]
Sawtooth1	164 922	789 014	R	[39]
SimCells	903 962	6 738 294	S	[18]
Cells	3 582 102	31 537 228	R	[18]

**Table 3: Summary of the linear programs used for evaluation.
All instances are from real problems.**

Name	Rows	Cols.	Non- zeros	Sol. time	Source
qap15	6 331	22 275	110 700	22 min	[27]
nug08-3rd	19 728	20 448	139 008	100 min	[27]
supportcase10	10 713	1 429 098	4 287 094	31 min	[27]
ex10	69 609	17 680	1 179 680	24 min	[27]

Experiments

Speed-Accuracy Tradeoff

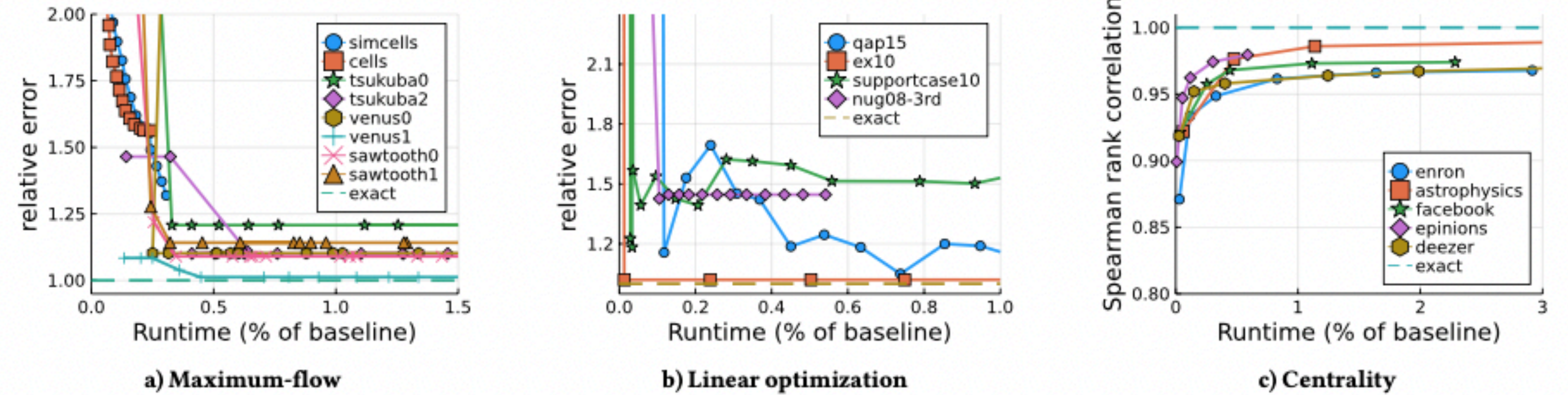


Figure 7: Speed-accuracy trade-offs for three task types and 20 datasets. Runtime reported is end-to-end, including the time taken for graph coloring, building an approximate instance of the problem and solving it.

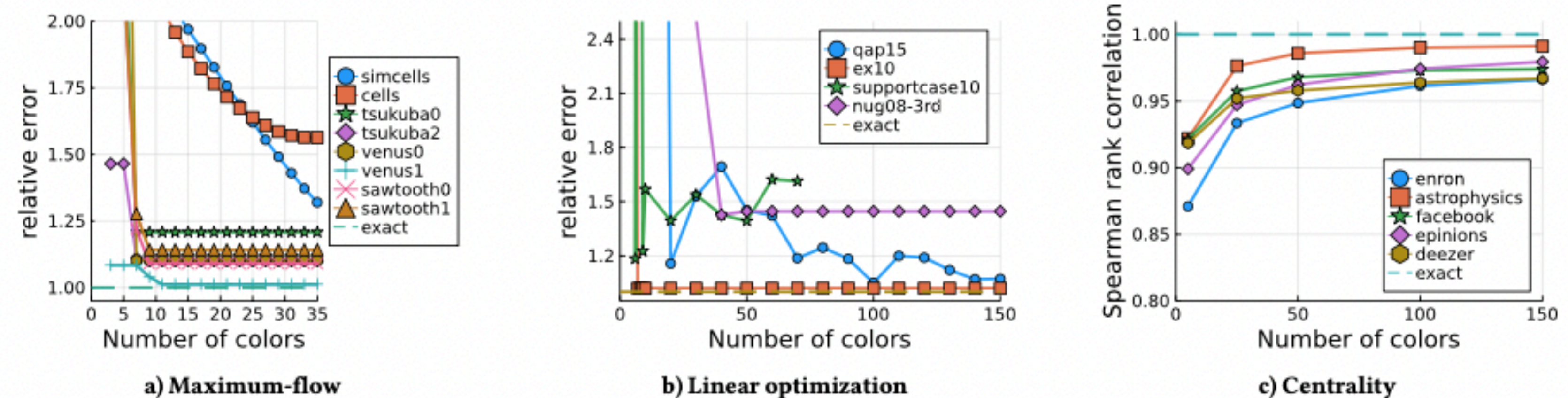


Figure 8: Accuracy as a function of the number of colors, across the same three tasks.

Experiments

Runtime and compression ratios

Table 4: Runtime and compression ratios of quasi-stable coloring vs. prior work (stable coloring [4, 22]) for selected datasets.

Dataset	Max q	Mean q	Colors	Compression	Time
OpenFlights	stable ($q=0$)		2 637	1.29:1	150ms
	$q = 64$	15.8	9	380:1	10ms
	$q = 32$	6.96	17	200:1	20ms
	$q = 16$	2.22	39	87:1	60ms
	$q = 8$	0.52	106	32:1	350ms
Epinions	stable ($q=0$)		53 068	1.42:1	49s
	$q = 64$	4.42	71	1 000:1	2.39s
	$q = 32$	1.17	144	526:1	8.95s
	$q = 16$	0.79	316	240:1	40.5s
	$q = 8$	0.22	869	87:1	5m19s
DBLP	stable ($q=0$)		233 466	1.35:1	14m52s
	$q = 64$	11.94	21	15 000:1	2.28s
	$q = 32$	2.22	89	3 500:1	22.6s
	$q = 16$	0.39	373	850:1	6m39s
	$q = 8$	0.06	1513	210:1	2h38m

Experiments

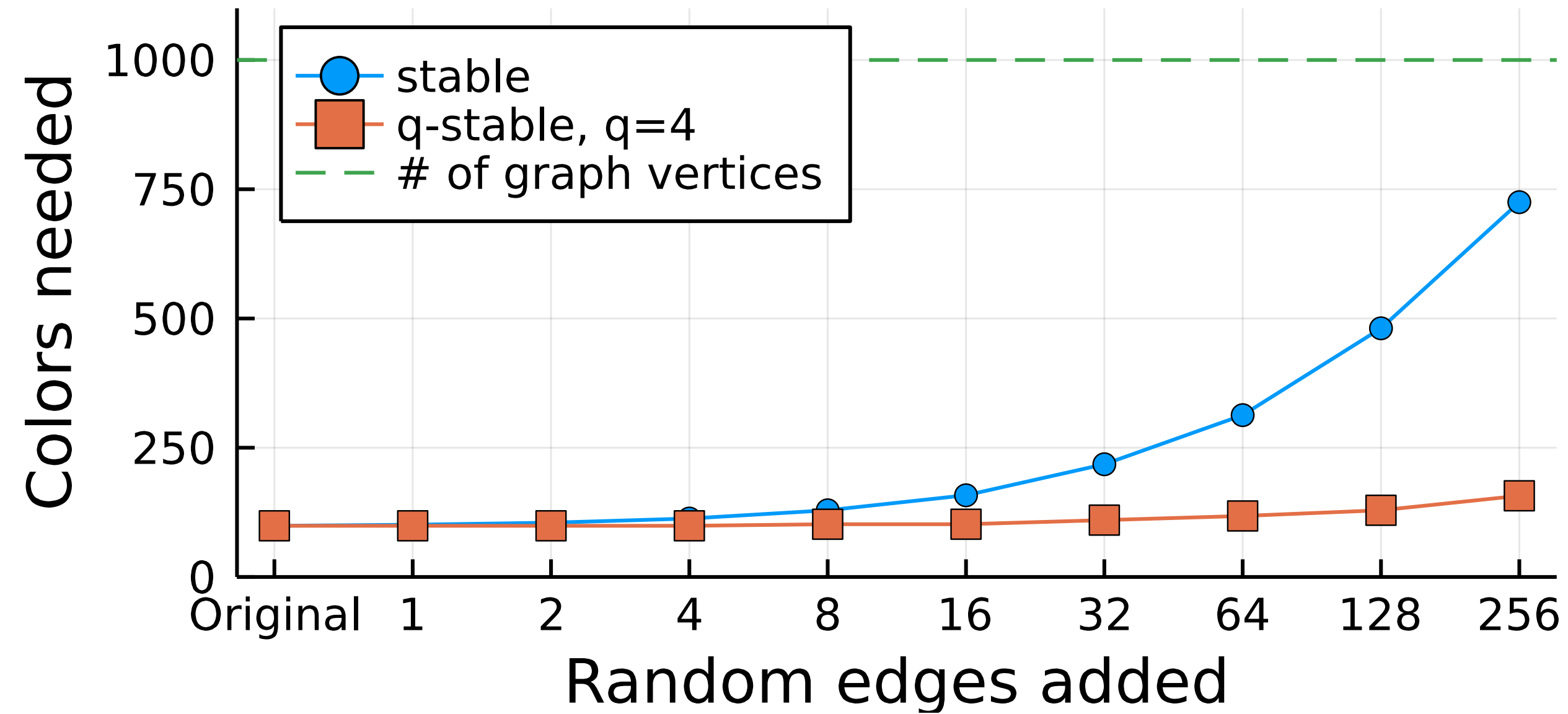
Q-Stable Coloring vs. Prior Approximations

Table 1: Runtime comparison of q-stable colors vs. prior approximations (Riondato-Kornaropoulos [37] and early-stopping [33]) and exact algorithms (Brandes [5] and interior-point solver [43]). Runtime to achieve a target approximation quality is measured; target is correlation (ρ) with ground truth values for centrality and relative error for linear optimization. “ \times ” is 20-minute timeout. Units in seconds, lower is better.

	Betweenness centrality: ours, [37], and [5]						Exact
	$\rho=0.90$		$\rho=0.95$		$\rho=0.97$		
	Ours	Prior	Ours	Prior	Ours	Prior	
Astroph.	0.13	15.2	1.03	41.4	2.49	61.1	223
Facebook	0.07	3.2	0.53	7.1	2.23	12.6	221
Deezer	0.05	3.6	1.11	7.2	8.56	14.8	295
Enron	0.41	2.6	3.06	5.6	10.8	8.7	380
Epinions	0.18	17.1	3.15	36.5	7.95	58.2	2552
	Linear optimization: ours, [33], and [43]						Exact
	rel. err.=3.0		rel. err.=2.0		rel. err.=1.5		
	Ours	Prior	Ours	Prior	Ours	Prior	
qap15	3.20	112.	4.91	524.	11.4	×	1 320
nug08.	5.40	1 027.	6.65	×	6.65	×	6 000
support.	0.51	143.	4.98	143.	×	×	1 860
ex10	0.247	795.	14.0	795.	14.0	795.	1 440

Quasi-Stable Colors

Robust against perturbations



Aside: Julia for Academic Research

The Good

- Excellent fit with academic work patterns: rapid development cycle when testing ideas
- Progressive improvement for performance-critical components
- Ecosystem tooling allows for easily distribution, replication of work

The Bad

- Decision paralysis: ambiguity in library choice confusing for newcomers
- Limitations in tooling (CPU and memory profiling) make performance issues opaque
- Language conventions, software design patterns still developing (risk of anti-patterns)

Recap

Reference implementation at
<https://github.com/mkyl/QuasiStableColors.jl>

Install the package:

```
moe@MacBook-Pro ~> julia  
(@v1.8) pkg> add QuasiStableColors
```

Got potential applications? Reach out!

kayali@cs.washington.edu
@moe_kayali

Recap

Reference implementation at
<https://github.com/mkyl/QuasiStableColors.jl>

Install the package:

```
moe@MacBook-Pro ~> julia  
(@v1.8) pkg> add QuasiStableColors
```

Got potential applications? Reach out!

kayali@cs.washington.edu
@moe_kayali

Questions?

Algorithm

Color refinement algorithm is not practical for q-coloring.

For stable coloring, all refinement orders are equivalent and arrive at the unique such color.

Not so for q-colors: choice of partitions to refine decides quality of coloring. Generally, many distinct colors for a fixed value of q .

Algorithm 1: Computing an approximate partition over weighted graph G , with n colors or ε maximum error

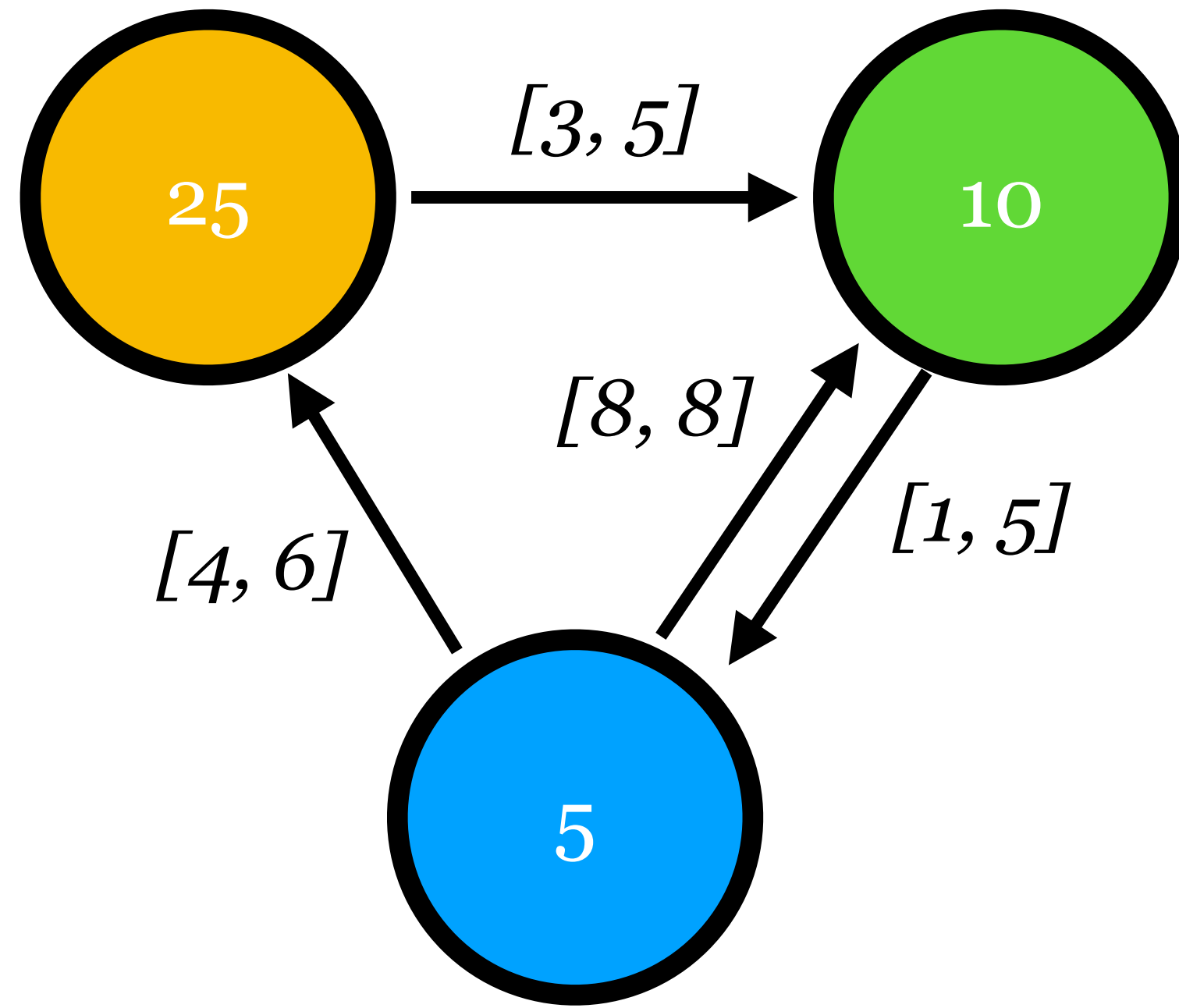
Data: $G = (V, E), W : V \times V \rightarrow \mathbb{R}^+$

Parameters: $n \in \mathbb{Z}^+, \varepsilon \in \mathbb{R}_{\geq 0}$

Result: $P \subset \mathcal{P}(V)$

```
1  $P \leftarrow \{V\};$ 
2 while  $|P| < n$  do
3    $U_{ij}, L_{ij} \leftarrow \max_{v \in P_i} \deg(v, P_j), \min_{v \in P_i} \deg(v, P_j);$ 
4    $Err \leftarrow U - L;$ 
5   if  $\max Err \leq \varepsilon$  then
6     break;
7    $C_{ij} \leftarrow |E \cap (P_i \times P_j)|;$  // count edges
8    $Err_{\text{weighted}} \leftarrow Err \odot C;$  // element-wise product
9    $i, j \leftarrow \operatorname{argmax}_{i,j} Err_{\text{weighted}};$  // witness
10   $\text{threshold} \leftarrow \sqrt{U_{ij} \times L_{ij}};$ 
11  // Split  $P_i$  at threshold
12   $P_{\text{retain}} \leftarrow \{v \in P_i \mid \deg(v, P_j) \leq \text{threshold}\};$ 
13   $P_{\text{eject}} \leftarrow P_i \setminus P_{\text{retain}};$ 
14   $P \leftarrow P \setminus \{P_i\} \cup \{P_{\text{retain}}, P_{\text{eject}}\};$ 
```

Algorithm



Quotient Graph
 $[min, max]$ degree

Algorithm 1: Computing an approximate partition over weighted graph G , with n colors or ε maximum error

Data: $G = (V, E), W : V \times V \rightarrow \mathbb{R}^+$

Parameters: $n \in \mathbb{Z}^+, \varepsilon \in \mathbb{R}_{\geq 0}$

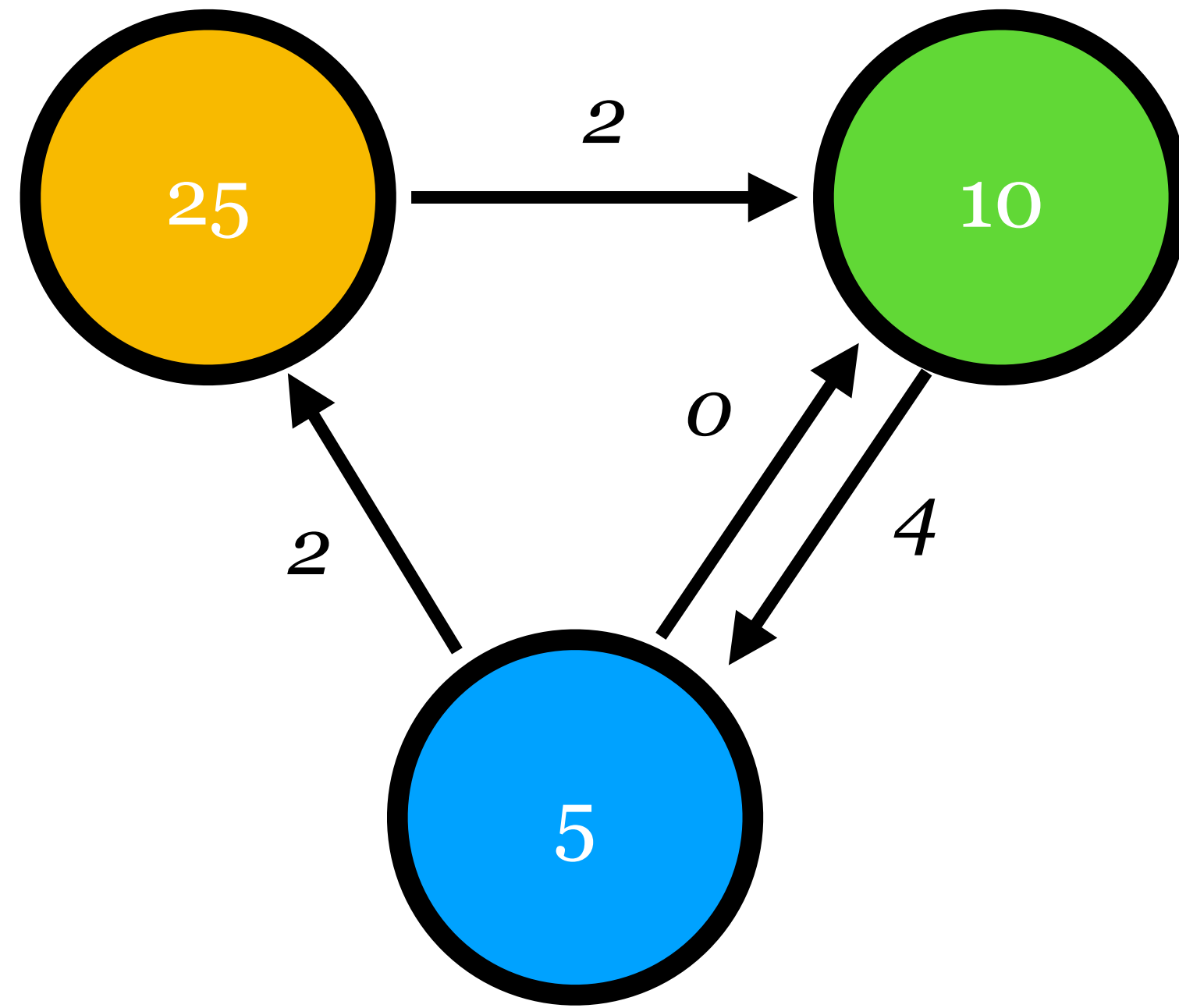
Result: $P \subset \mathcal{P}(V)$

```

1  $P \leftarrow \{V\};$ 
2 while  $|P| < n$  do
3    $U_{ij}, L_{ij} \leftarrow \max_{v \in P_i} \deg(v, P_j), \min_{v \in P_i} \deg(v, P_j);$ 
4    $Err \leftarrow U - L;$ 
5   if  $\max Err \leq \varepsilon$  then
6     break;
7    $C_{ij} \leftarrow |E \cap (P_i \times P_j)|;$  // count edges
8    $Err_{\text{weighted}} \leftarrow Err \odot C;$  // element-wise product
9    $i, j \leftarrow \operatorname{argmax}_{i,j} Err_{\text{weighted}};$  // witness
10   $\text{threshold} \leftarrow \sqrt{U_{ij} \times L_{ij}};$ 
11  // Split  $P_i$  at threshold
12   $P_{\text{retain}} \leftarrow \{v \in P_i \mid \deg(v, P_j) \leq \text{threshold}\};$ 
13   $P_{\text{eject}} \leftarrow P_i \setminus P_{\text{retain}};$ 
14   $P \leftarrow P \setminus \{P_i\} \cup \{P_{\text{retain}}, P_{\text{eject}}\};$ 

```

Algorithm



Quotient Graph
degree range

Algorithm 1: Computing an approximate partition over weighted graph G , with n colors or ε maximum error

Data: $G = (V, E), W : V \times V \rightarrow \mathbb{R}^+$

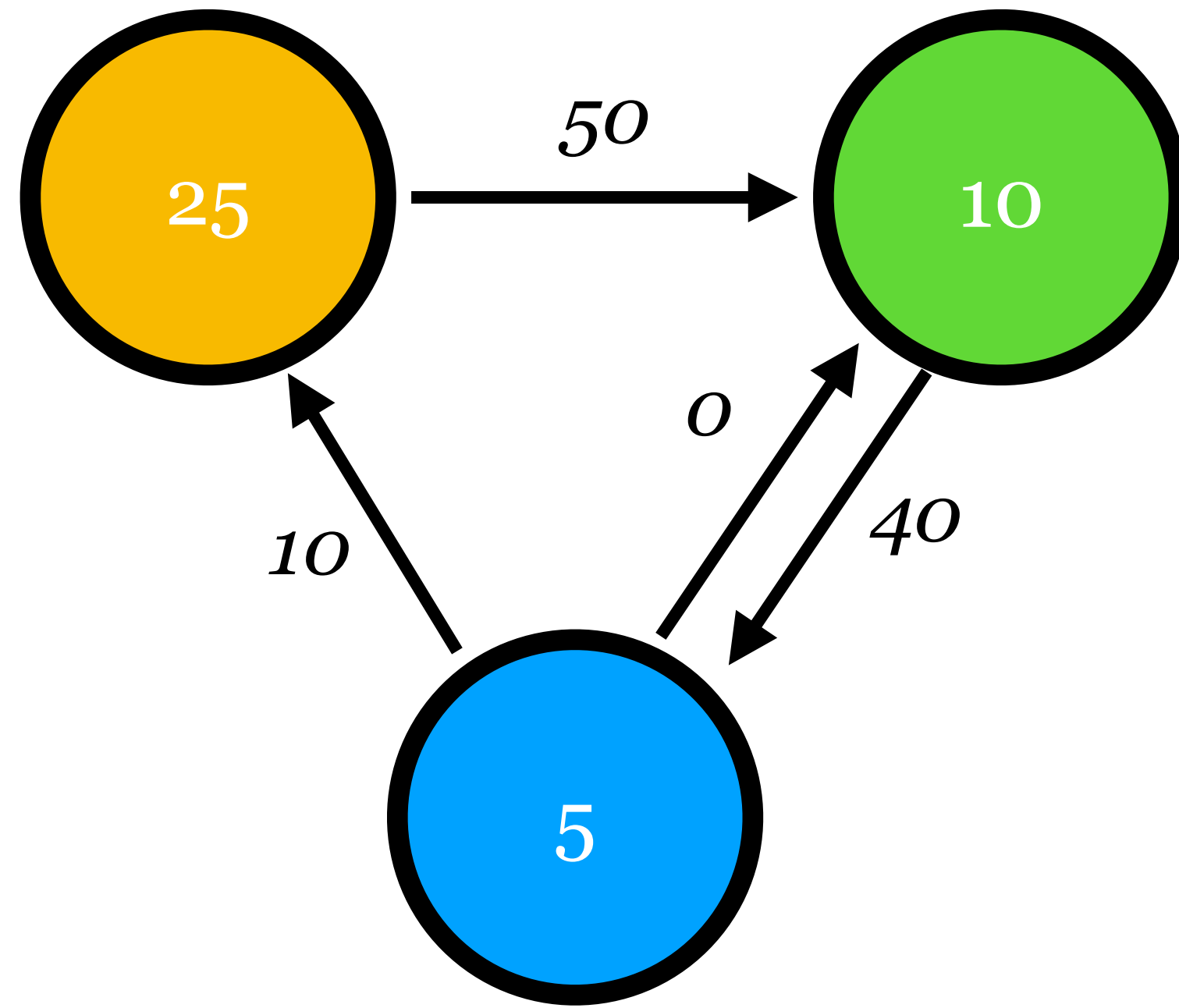
Parameters: $n \in \mathbb{Z}^+, \varepsilon \in \mathbb{R}_{\geq 0}$

Result: $P \subset \mathcal{P}(V)$

```

1  $P \leftarrow \{V\};$ 
2 while  $|P| < n$  do
3    $U_{ij}, L_{ij} \leftarrow \max_{v \in P_i} \deg(v, P_j), \min_{v \in P_i} \deg(v, P_j);$ 
4    $Err \leftarrow U - L;$ 
5   if  $\max Err \leq \varepsilon$  then
6     break;
7    $C_{ij} \leftarrow |E \cap (P_i \times P_j)|;$            // count edges
8    $Err_{\text{weighted}} \leftarrow Err \odot C;$          // element-wise product
9    $i, j \leftarrow \operatorname{argmax}_{i,j} Err_{\text{weighted}};$  // witness
10   $\text{threshold} \leftarrow \sqrt{U_{ij} \times L_{ij}};$ 
11  // Split  $P_i$  at threshold
12   $P_{\text{retain}} \leftarrow \{v \in P_i \mid \deg(v, P_j) \leq \text{threshold}\};$ 
13   $P_{\text{eject}} \leftarrow P_i \setminus P_{\text{retain}};$ 
14   $P \leftarrow P \setminus \{P_i\} \cup \{P_{\text{retain}}, P_{\text{eject}}\};$ 
  
```

Algorithm



Quotient Graph
(weighted) degree range

Algorithm 1: Computing an approximate partition over weighted graph G , with n colors or ε maximum error

Data: $G = (V, E), W : V \times V \rightarrow \mathbb{R}^+$

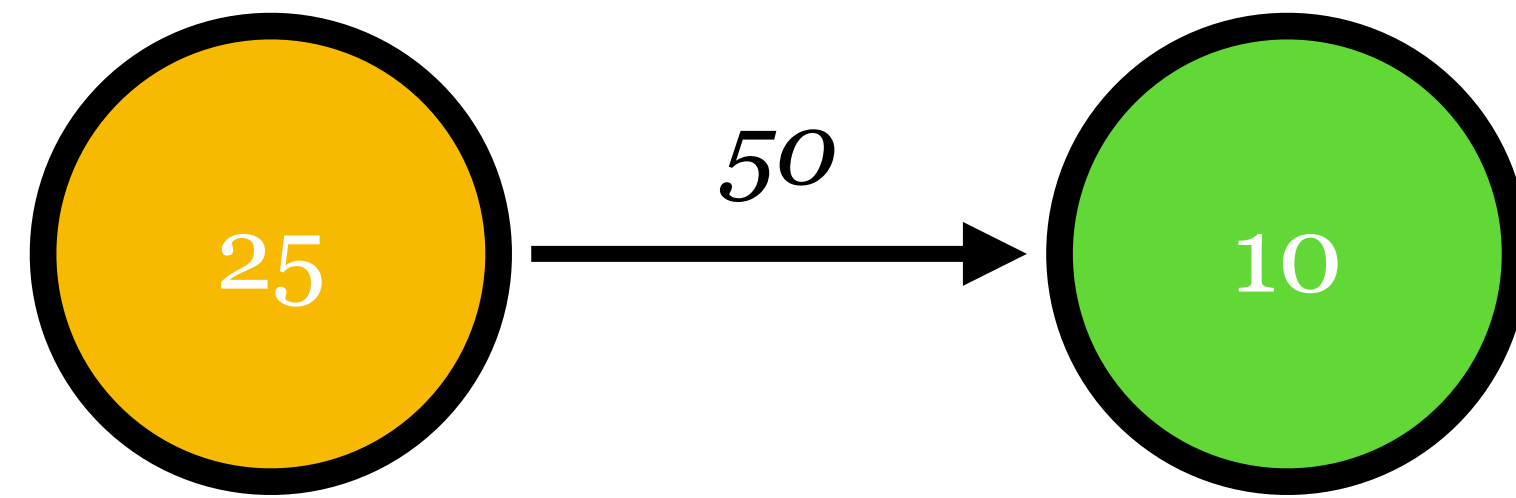
Parameters: $n \in \mathbb{Z}^+, \varepsilon \in \mathbb{R}_{\geq 0}$

Result: $P \subset \mathcal{P}(V)$

```

1  $P \leftarrow \{V\};$ 
2 while  $|P| < n$  do
3    $U_{ij}, L_{ij} \leftarrow \max_{v \in P_i} \deg(v, P_j), \min_{v \in P_i} \deg(v, P_j);$ 
4    $Err \leftarrow U - L;$ 
5   if  $\max Err \leq \varepsilon$  then
6     break;
7    $C_{ij} \leftarrow |E \cap (P_i \times P_j)|;$  // count edges
8    $Err_{\text{weighted}} \leftarrow Err \odot C;$  // element-wise product
9    $i, j \leftarrow \operatorname{argmax}_{i,j} Err_{\text{weighted}};$  // witness
10   $\text{threshold} \leftarrow \sqrt{U_{ij} \times L_{ij}};$ 
    // Split  $P_i$  at threshold
11   $P_{\text{retain}} \leftarrow \{v \in P_i \mid \deg(v, P_j) \leq \text{threshold}\};$ 
12   $P_{\text{eject}} \leftarrow P_i \setminus P_{\text{retain}};$ 
13   $P \leftarrow P \setminus \{P_i\} \cup \{P_{\text{retain}}, P_{\text{eject}}\};$ 
  
```

Algorithm



Quotient Graph
witness selection

Algorithm 1: Computing an approximate partition over weighted graph G , with n colors or ε maximum error

Data: $G = (V, E), W : V \times V \rightarrow \mathbb{R}^+$

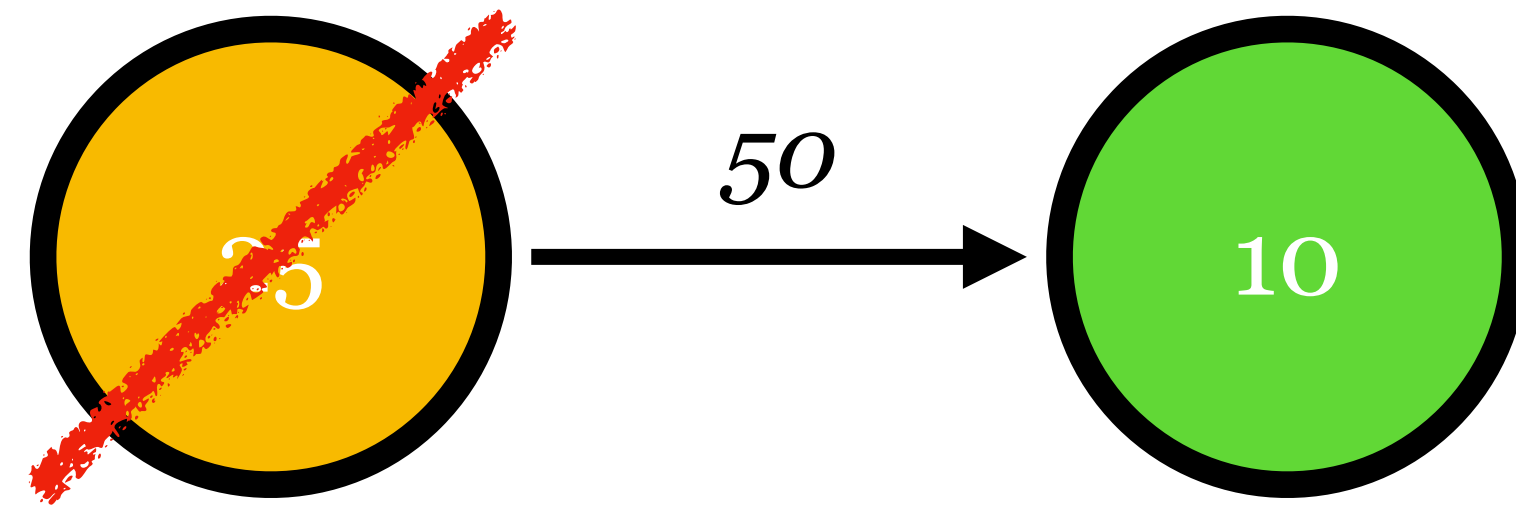
Parameters: $n \in \mathbb{Z}^+, \varepsilon \in \mathbb{R}_{\geq 0}$

Result: $P \subset \mathcal{P}(V)$

```

1  $P \leftarrow \{V\};$ 
2 while  $|P| < n$  do
3    $U_{ij}, L_{ij} \leftarrow \max_{v \in P_i} \deg(v, P_j), \min_{v \in P_i} \deg(v, P_j);$ 
4    $Err \leftarrow U - L;$ 
5   if  $\max Err \leq \varepsilon$  then
6     break;
7    $C_{ij} \leftarrow |E \cap (P_i \times P_j)|;$            // count edges
8    $Err_{\text{weighted}} \leftarrow Err \odot C;$          // element-wise product
9    $i, j \leftarrow \operatorname{argmax}_{i,j} Err_{\text{weighted}};$  // witness
10   $\text{threshold} \leftarrow \sqrt{U_{ij} \times L_{ij}};$ 
11  // Split  $P_i$  at threshold
12   $P_{\text{retain}} \leftarrow \{v \in P_i \mid \deg(v, P_j) \leq \text{threshold}\};$ 
13   $P_{\text{eject}} \leftarrow P_i \setminus P_{\text{retain}};$ 
14   $P \leftarrow P \setminus \{P_i\} \cup \{P_{\text{retain}}, P_{\text{eject}}\};$ 
  
```

Algorithm



Quotient Graph
witness selection

Algorithm 1: Computing an approximate partition over weighted graph G , with n colors or ε maximum error

Data: $G = (V, E), W : V \times V \rightarrow \mathbb{R}^+$

Parameters: $n \in \mathbb{Z}^+, \varepsilon \in \mathbb{R}_{\geq 0}$

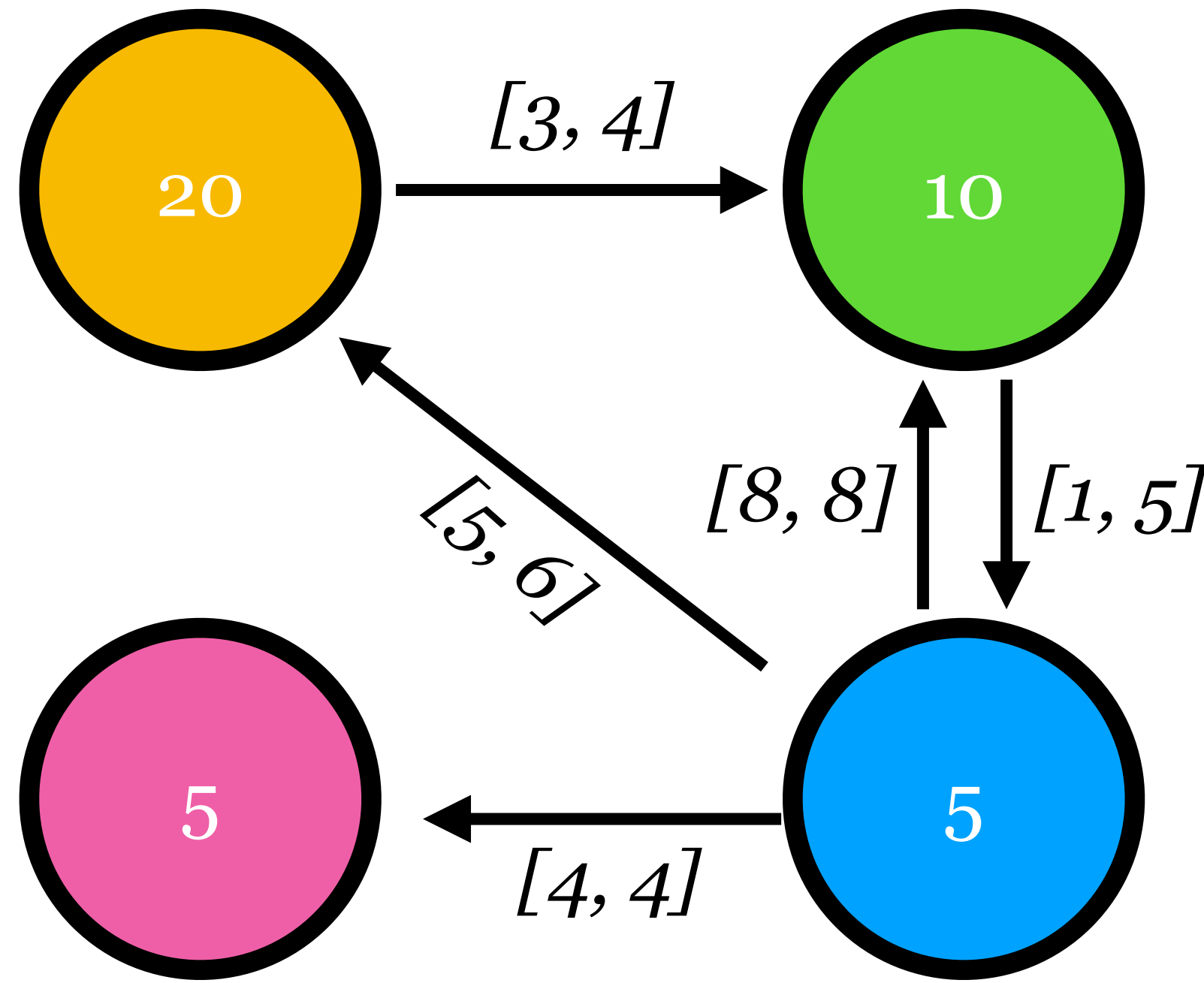
Result: $P \subset \mathcal{P}(V)$

```

1  $P \leftarrow \{V\};$ 
2 while  $|P| < n$  do
3    $U_{ij}, L_{ij} \leftarrow \max_{v \in P_i} \deg(v, P_j), \min_{v \in P_i} \deg(v, P_j);$ 
4    $Err \leftarrow U - L;$ 
5   if  $\max Err \leq \varepsilon$  then
6     break;
7    $C_{ij} \leftarrow |E \cap (P_i \times P_j)|;$  // count edges
8    $Err_{\text{weighted}} \leftarrow Err \odot C;$  // element-wise product
9    $i, j \leftarrow \operatorname{argmax}_{i,j} Err_{\text{weighted}};$  // witness
10   $\text{threshold} \leftarrow \sqrt{U_{ij} \times L_{ij}};$ 
    // Split  $P_i$  at threshold
11   $P_{\text{retain}} \leftarrow \{v \in P_i \mid \deg(v, P_j) \leq \text{threshold}\};$ 
12   $P_{\text{eject}} \leftarrow P_i \setminus P_{\text{retain}};$ 
13   $P \leftarrow P \setminus \{P_i\} \cup \{P_{\text{retain}}, P_{\text{eject}}\};$ 

```

Algorithm



Quotient Graph
iteration complete

Algorithm 1: Computing an approximate partition over weighted graph G , with n colors or ε maximum error

Data: $G = (V, E), W : V \times V \rightarrow \mathbb{R}^+$

Parameters: $n \in \mathbb{Z}^+, \varepsilon \in \mathbb{R}_{\geq 0}$

Result: $P \subset \mathcal{P}(V)$

```

1  $P \leftarrow \{V\};$ 
2 while  $|P| < n$  do
3    $U_{ij}, L_{ij} \leftarrow \max_{v \in P_i} \deg(v, P_j), \min_{v \in P_i} \deg(v, P_j);$ 
4    $Err \leftarrow U - L;$ 
5   if  $\max Err \leq \varepsilon$  then
6     break;
7    $C_{ij} \leftarrow |E \cap (P_i \times P_j)|;$  // count edges
8    $Err_{\text{weighted}} \leftarrow Err \odot C;$  // element-wise product
9    $i, j \leftarrow \operatorname{argmax}_{i,j} Err_{\text{weighted}};$  // witness
10   $\text{threshold} \leftarrow \sqrt{U_{ij} \times L_{ij}};$ 
11  // Split  $P_i$  at threshold
12   $P_{\text{retain}} \leftarrow \{v \in P_i \mid \deg(v, P_j) \leq \text{threshold}\};$ 
13   $P_{\text{eject}} \leftarrow P_i \setminus P_{\text{retain}};$ 
14   $P \leftarrow P \setminus \{P_i\} \cup \{P_{\text{retain}}, P_{\text{eject}}\};$ 

```

Color Refinement

aka 1-dimensional Weisfeiler-Leman

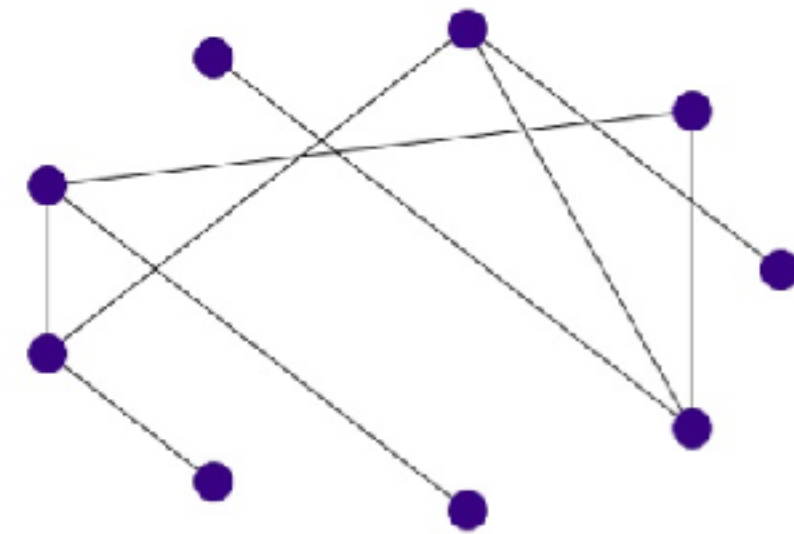
Color Refinement

aka 1-dimensional Weisfeiler-Leman

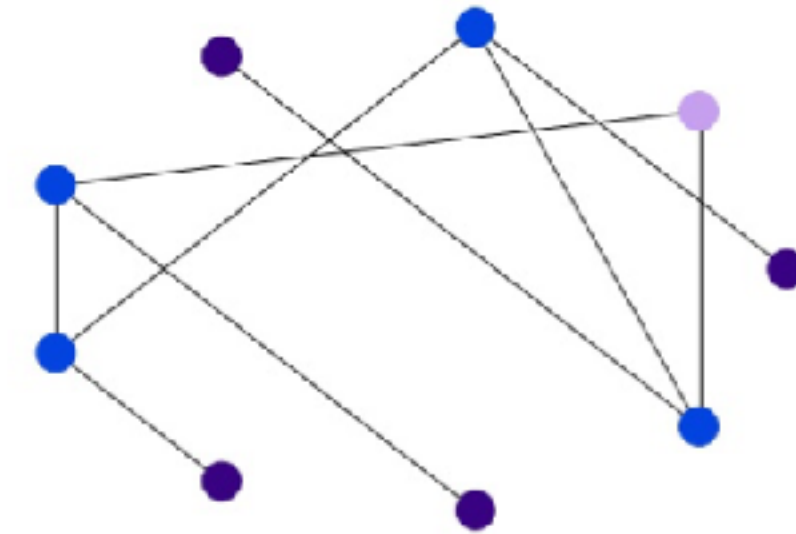
Note: *Distinct from proper vertex coloring, chromatic number, etc.*

Color Refinement

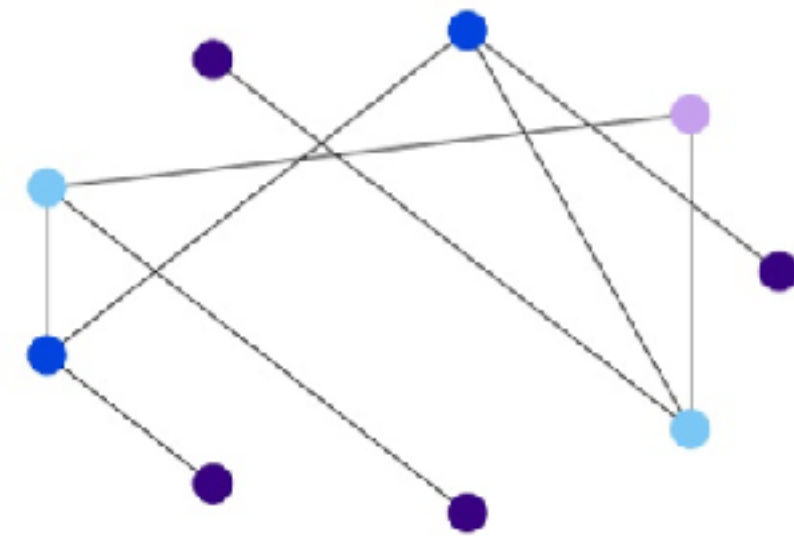
aka *1-dimensional Weisfeiler-Leman*



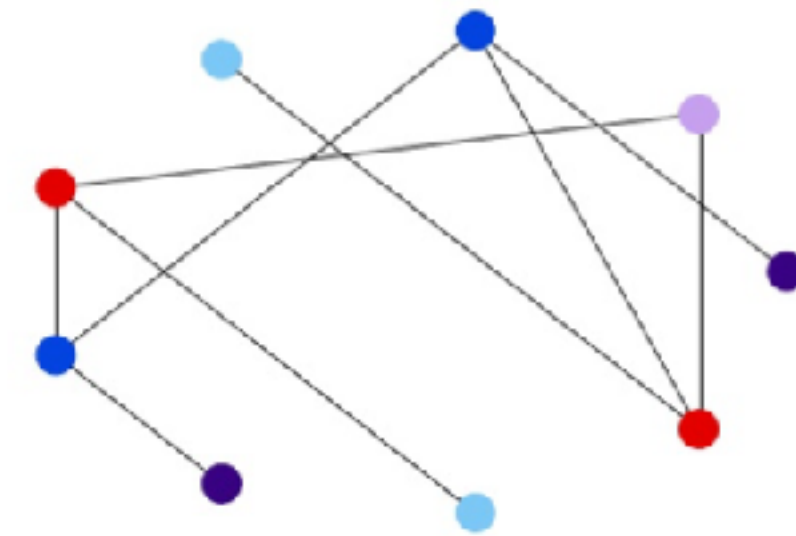
initial graph



colouring after round 1



colouring after round 2



stable colouring after round 3

Note: Distinct from proper vertex coloring, chromatic number, etc.

M. Grohe. word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data. PODS 2020, Portland, OR, USA, June 14-19, 2020, pages 1–16. ACM, 2020.

Color Refinement

DEFINITION. (*Stable Coloring*) Call a coloring $P = \{P_1, \dots, P_k\}$ stable if

$$\forall i, j, \forall x, y \in P_i : |N(x) \cap P_j| = |N(y) \cap P_j|$$

where $N(v)$ denotes the neighbors of vertex v .