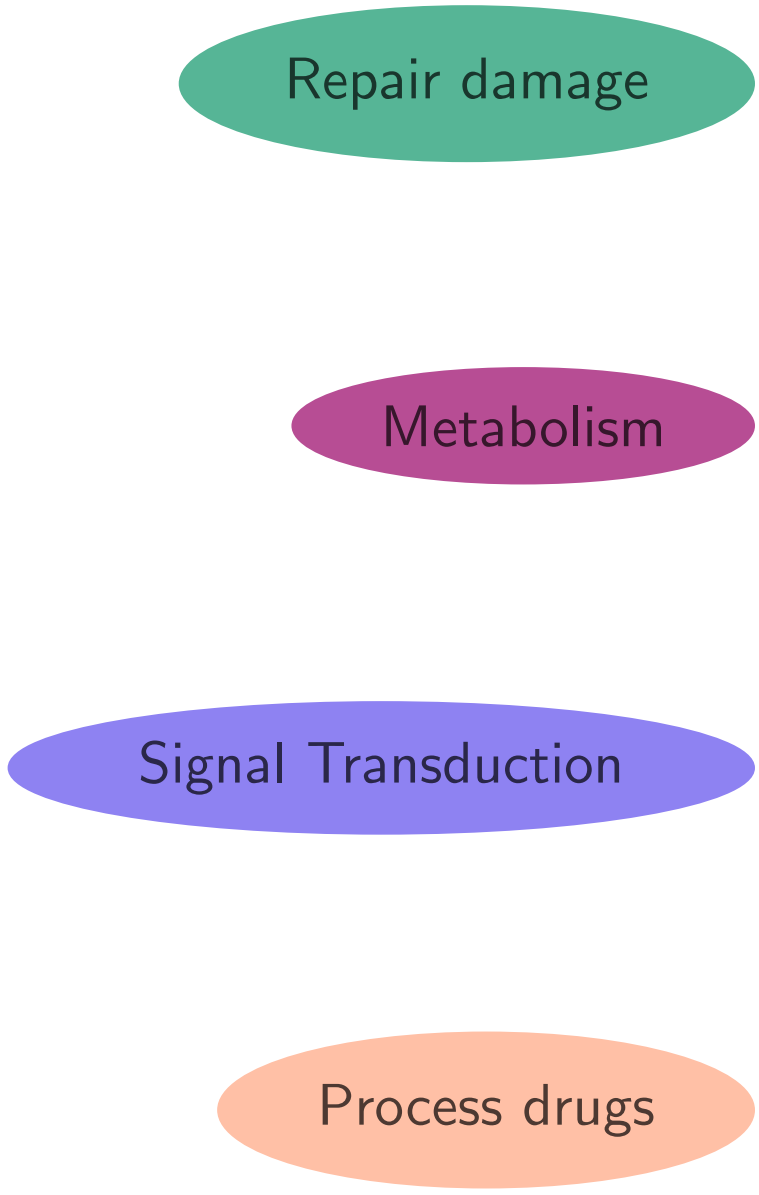
Gene Module Decomposition

Madison Cooley
Casey S. Greene, Davis Issac, Milton Pividori, Blair D. Sullivan

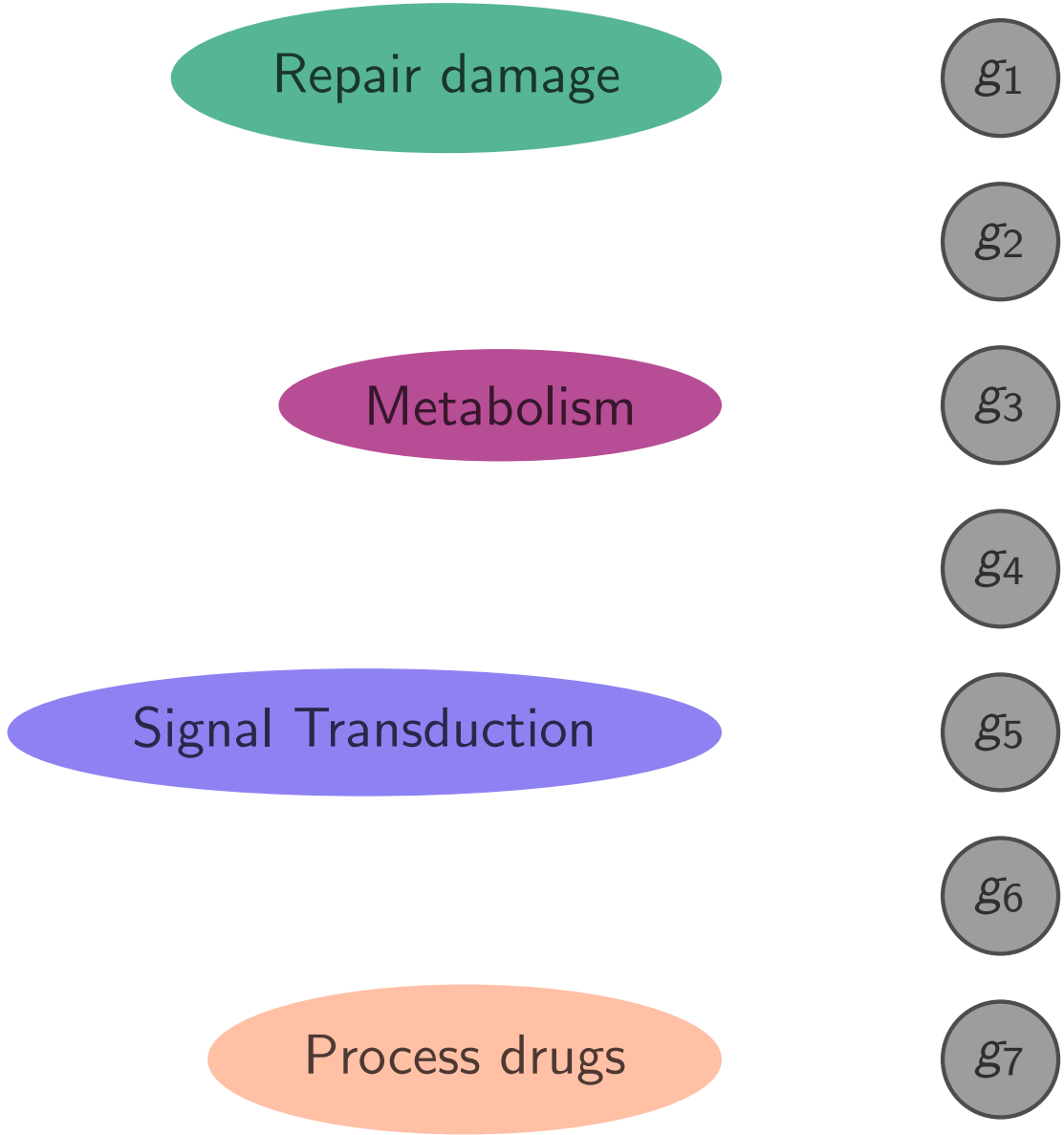
Parameterized algorithms for identifying gene co-expression modules via weighted clique decomposition

University of Utah | mcooley@cs.utah.edu

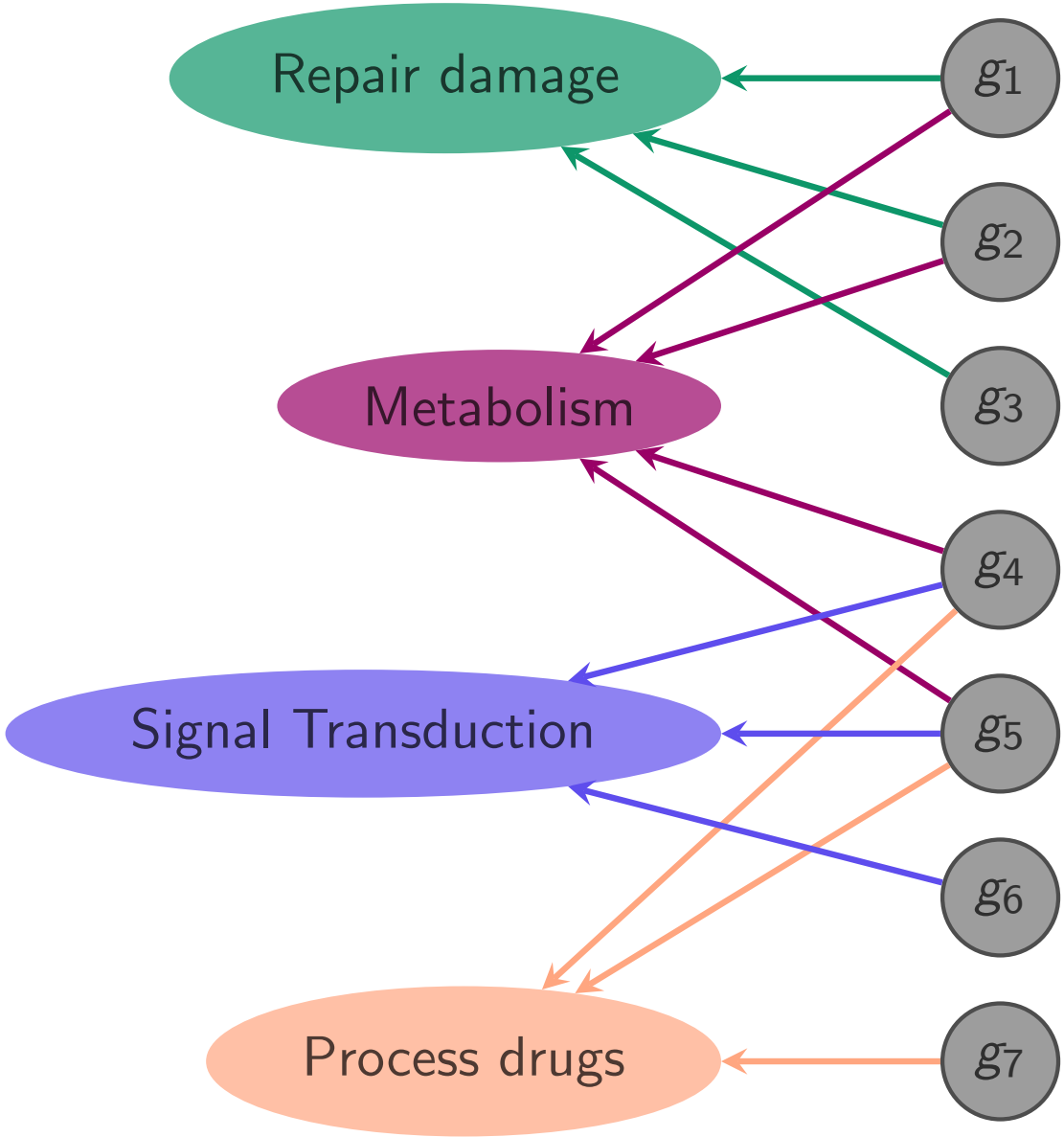
Biological Modules



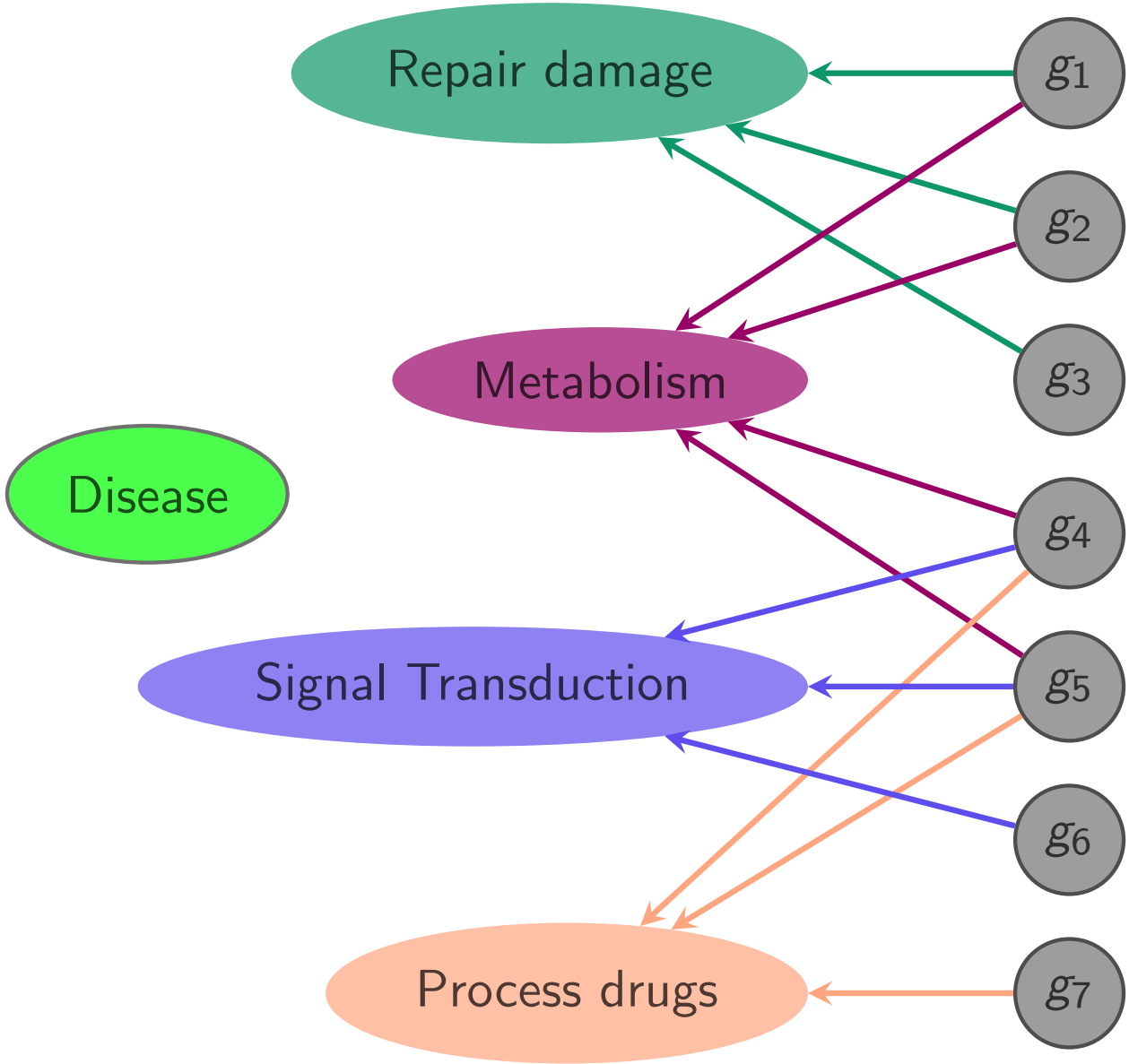
Biological Modules



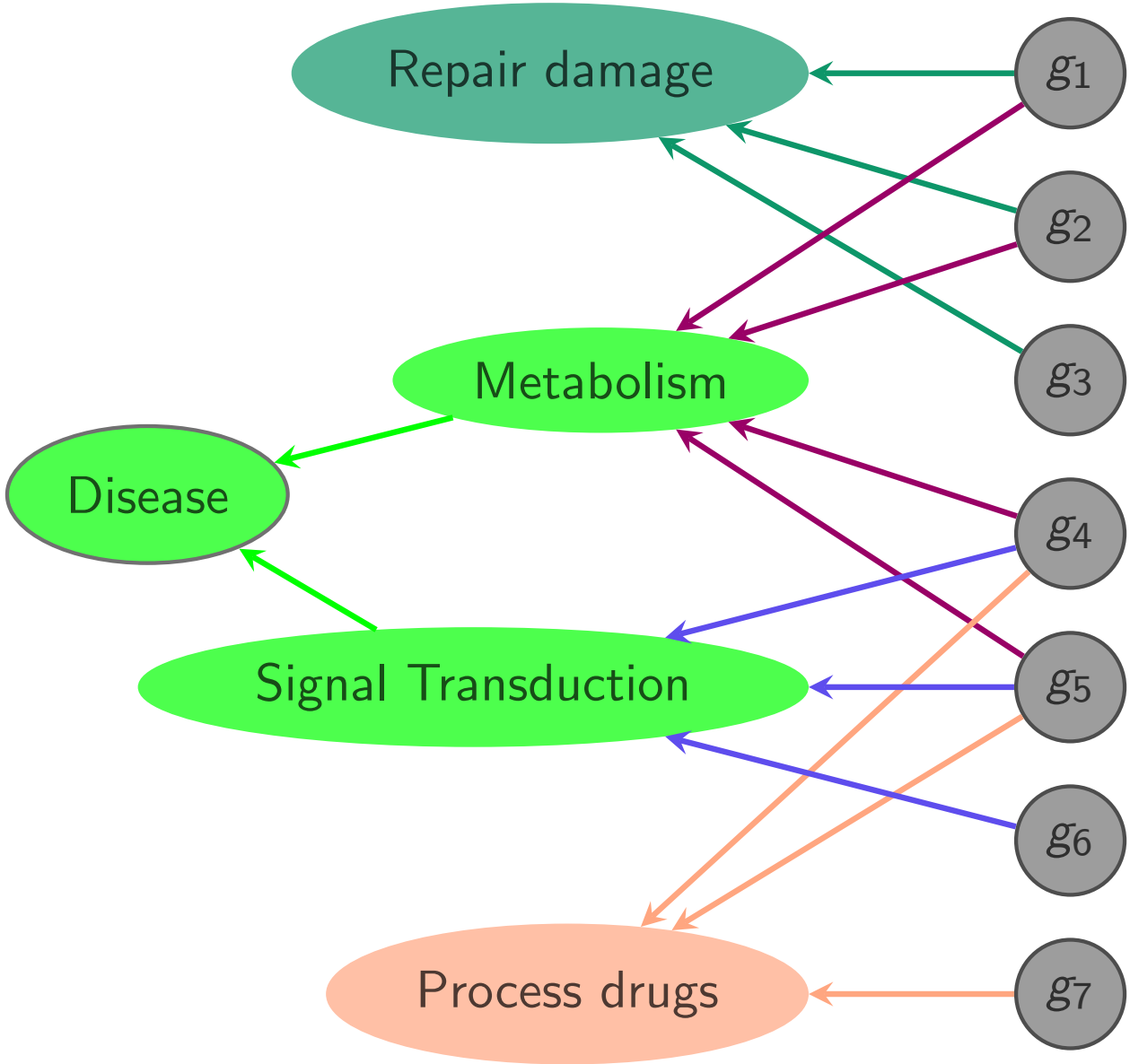
Biological Modules



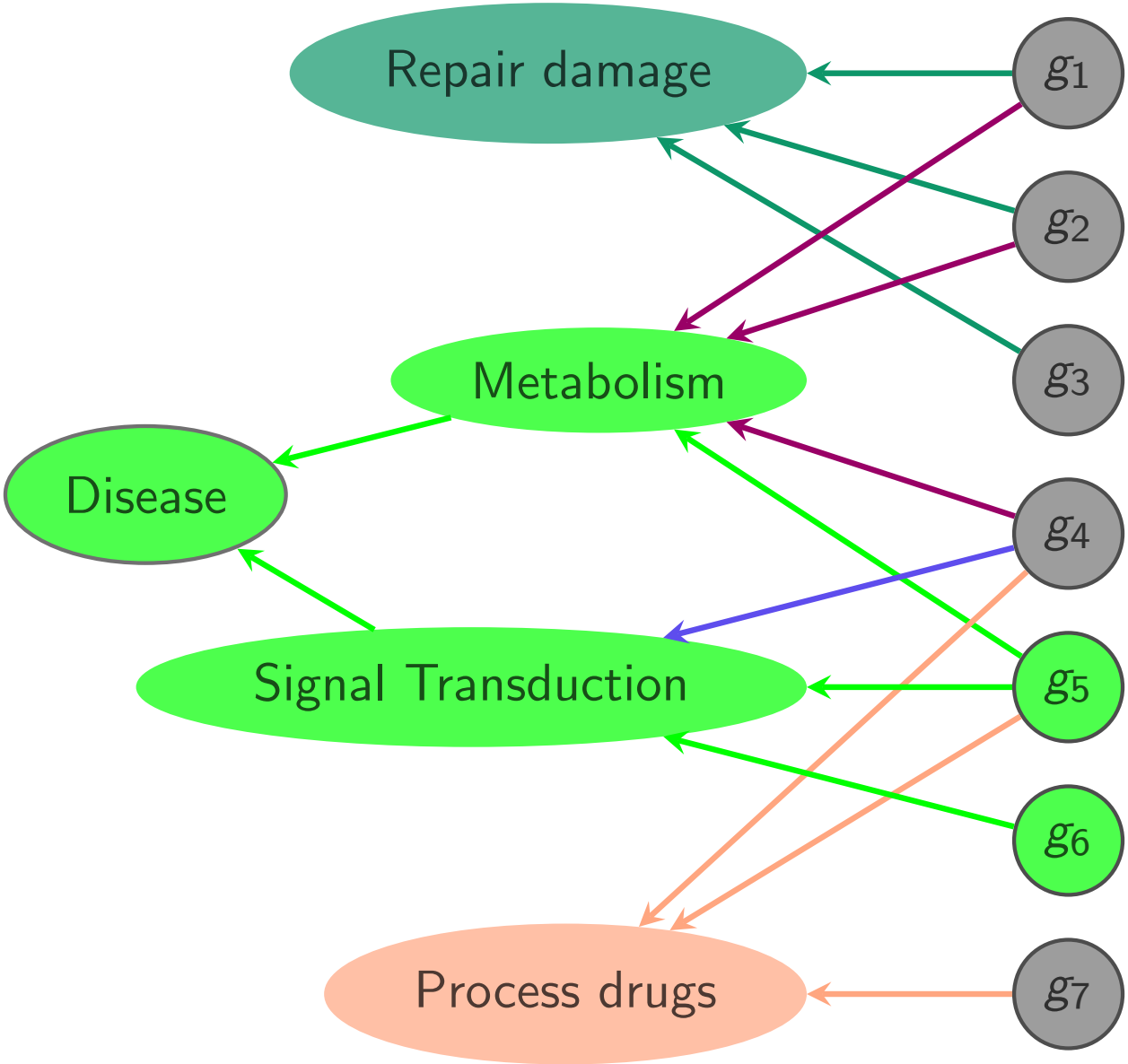
Biological Modules



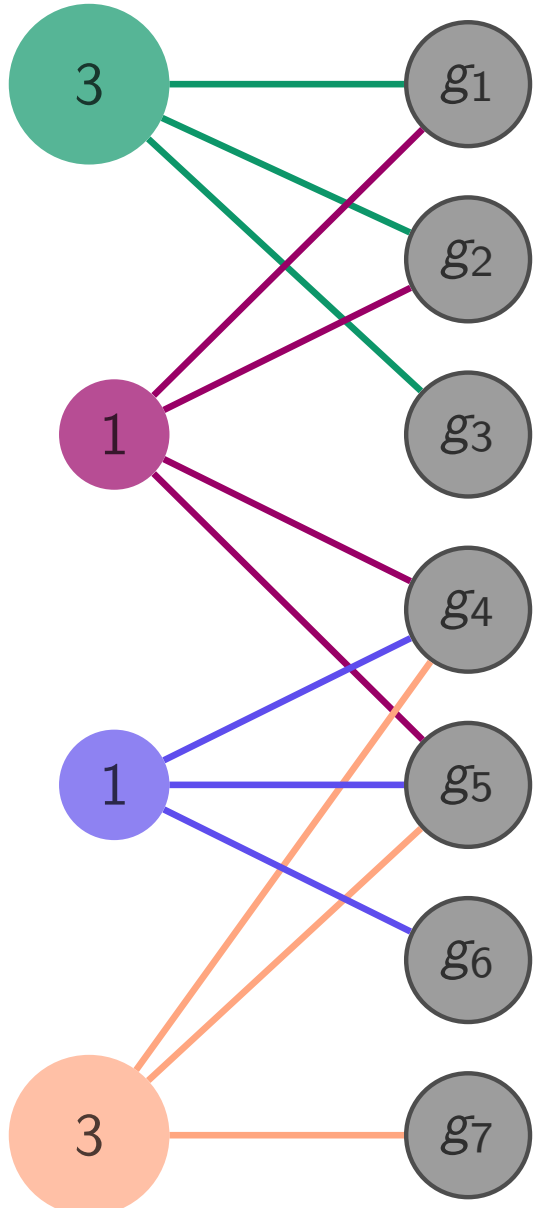
Biological Modules



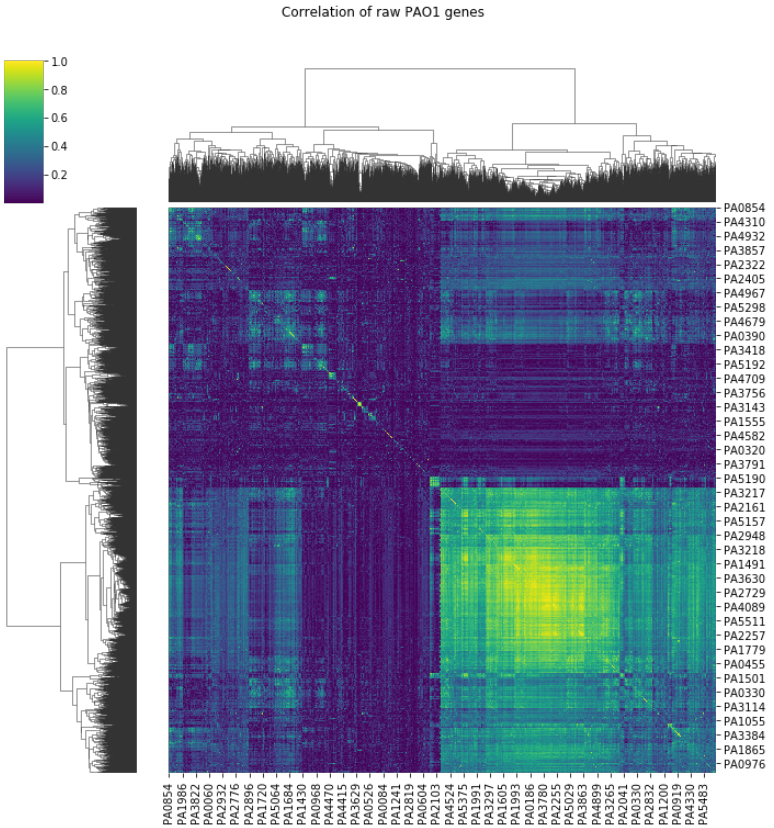
Biological Modules

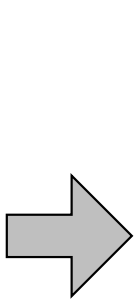


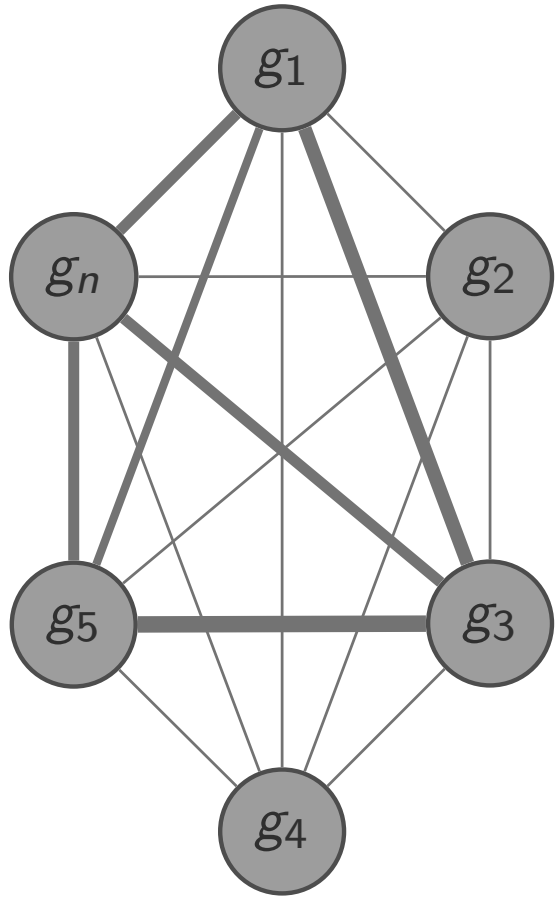
Problem Modeling



Gene Co-Expression Networks[1]

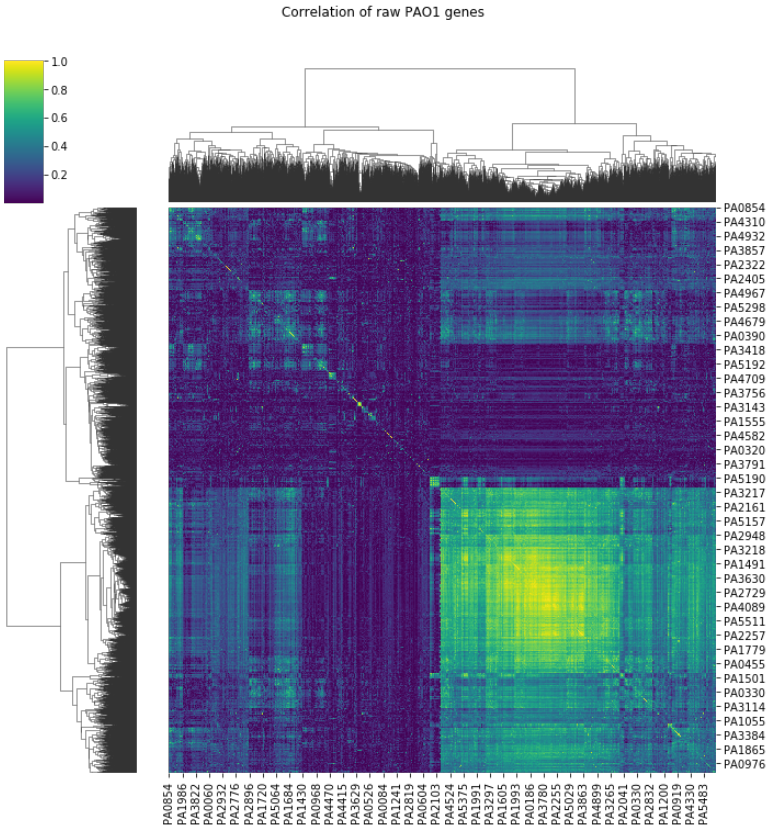


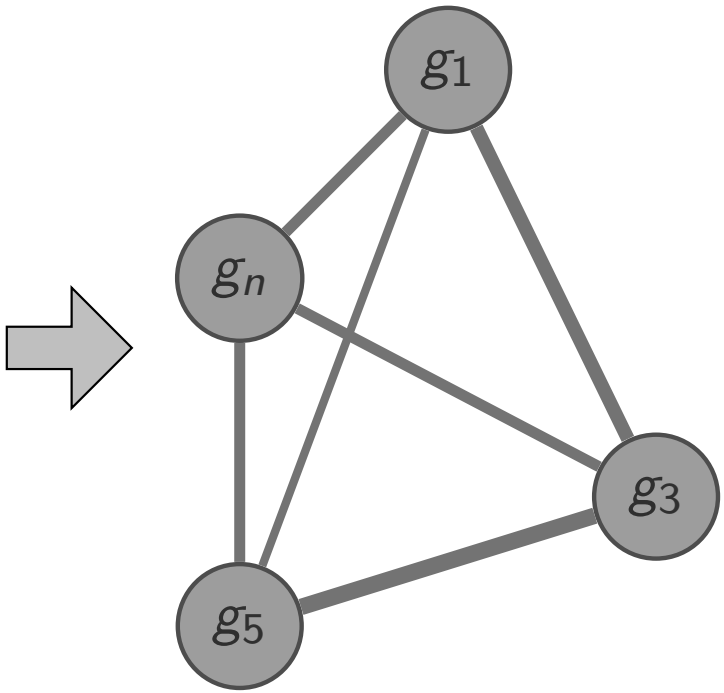




[1] Image from Alexandra Lee, PhD Student, Greene Laboratory

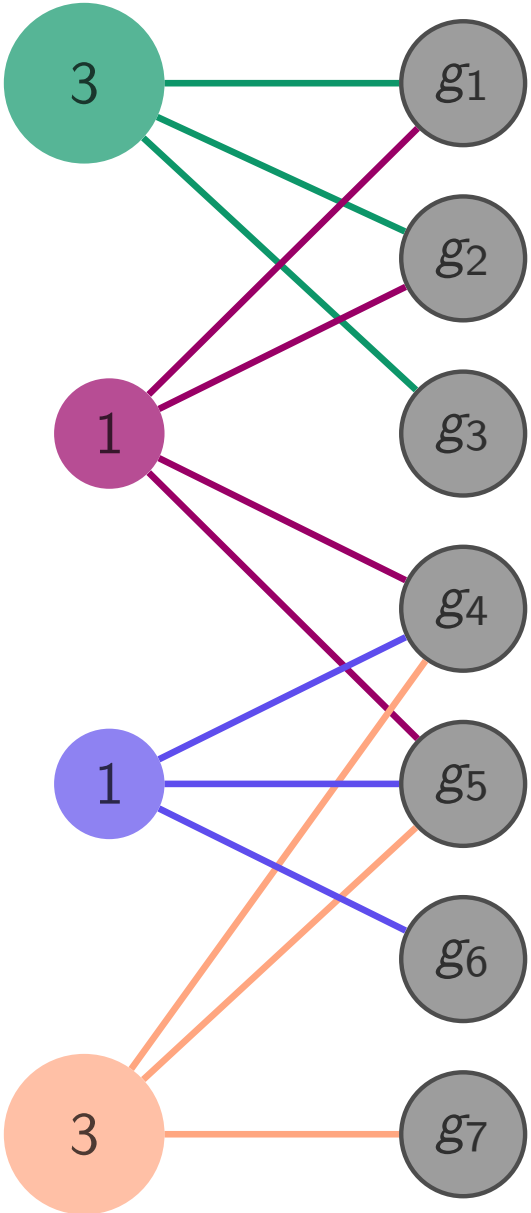
Gene Co-Expression Networks[1]



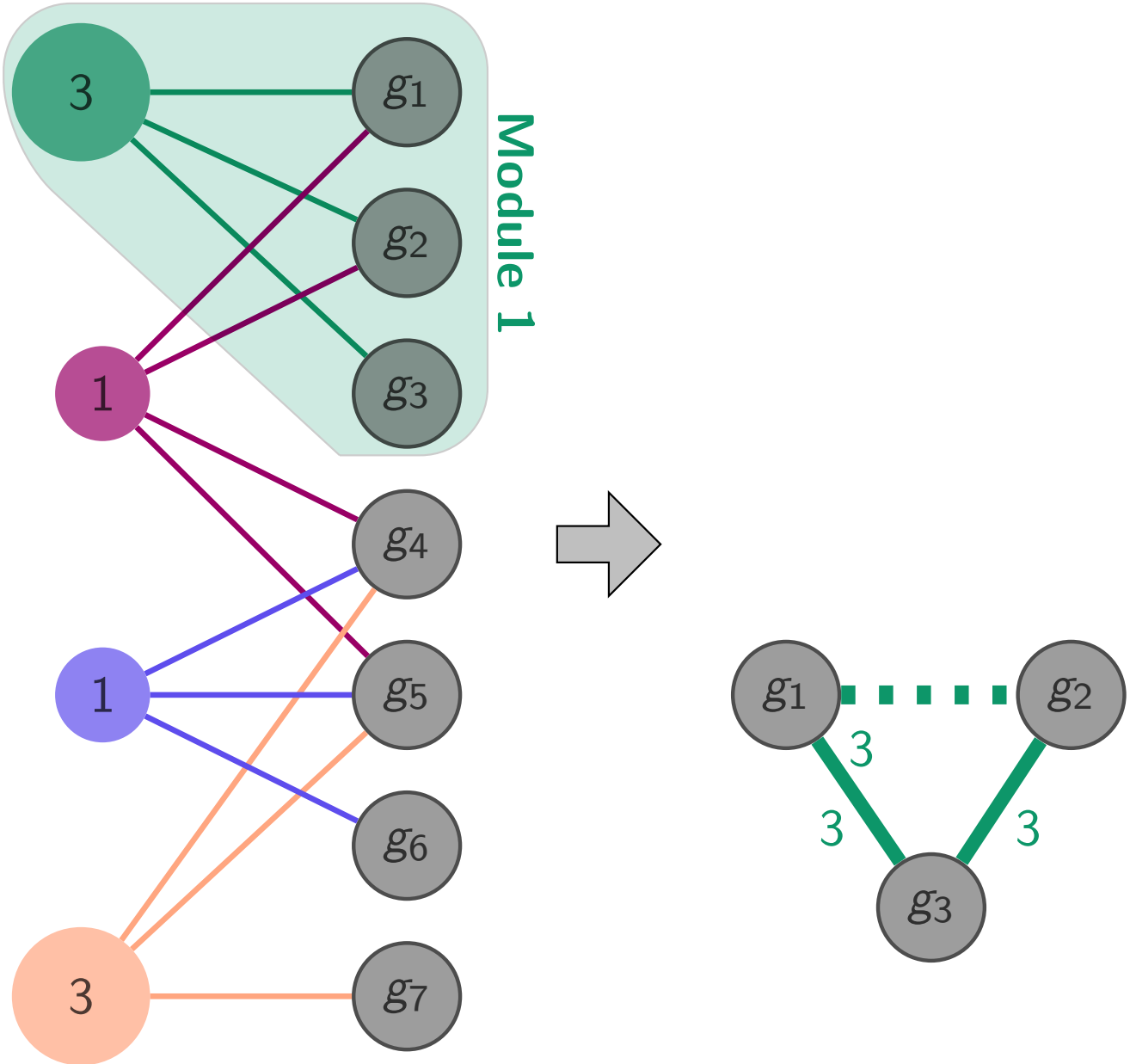


[1] Image from Alexandra Lee, PhD Student, Greene Laboratory

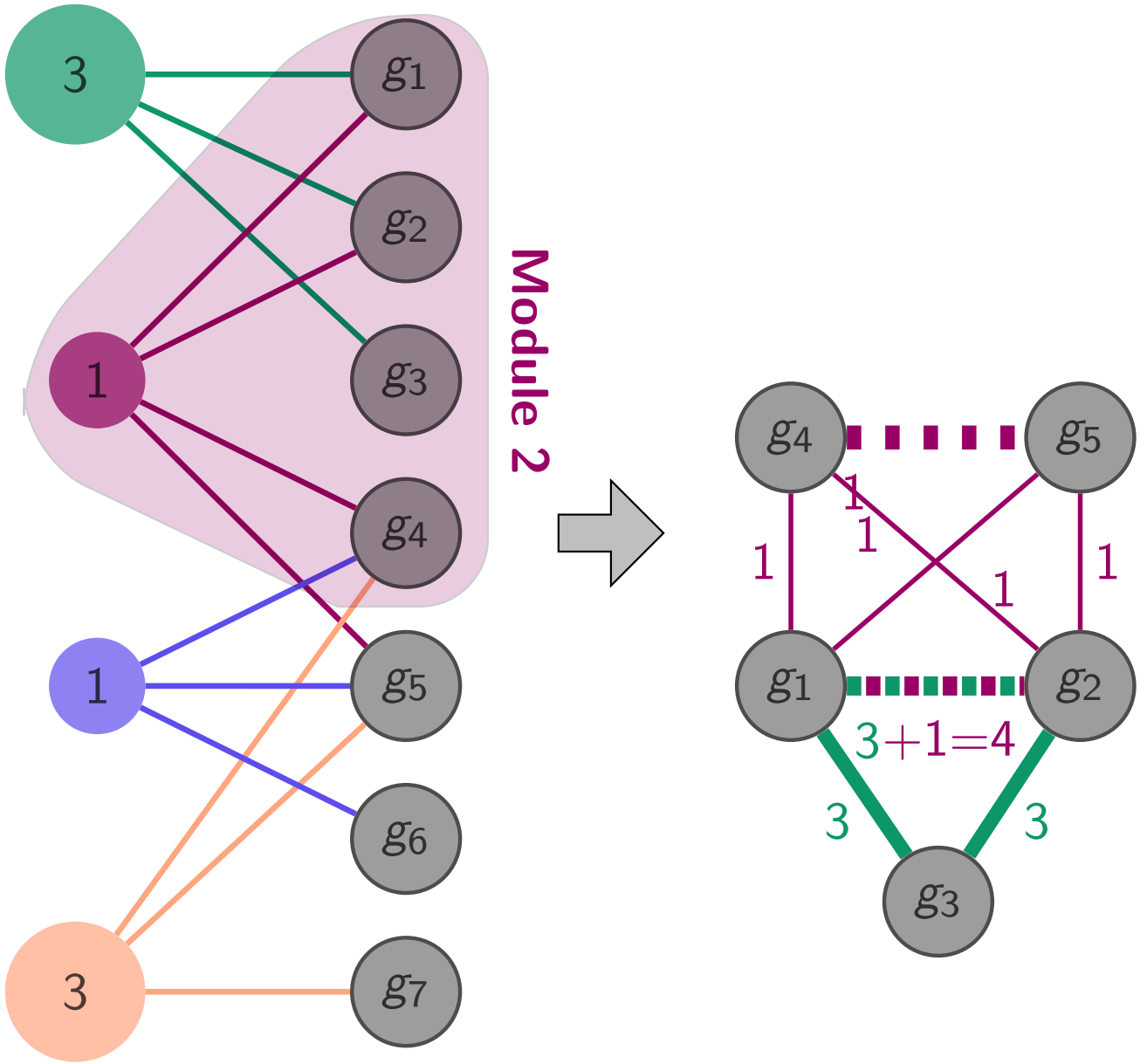
Gene-Gene Projection



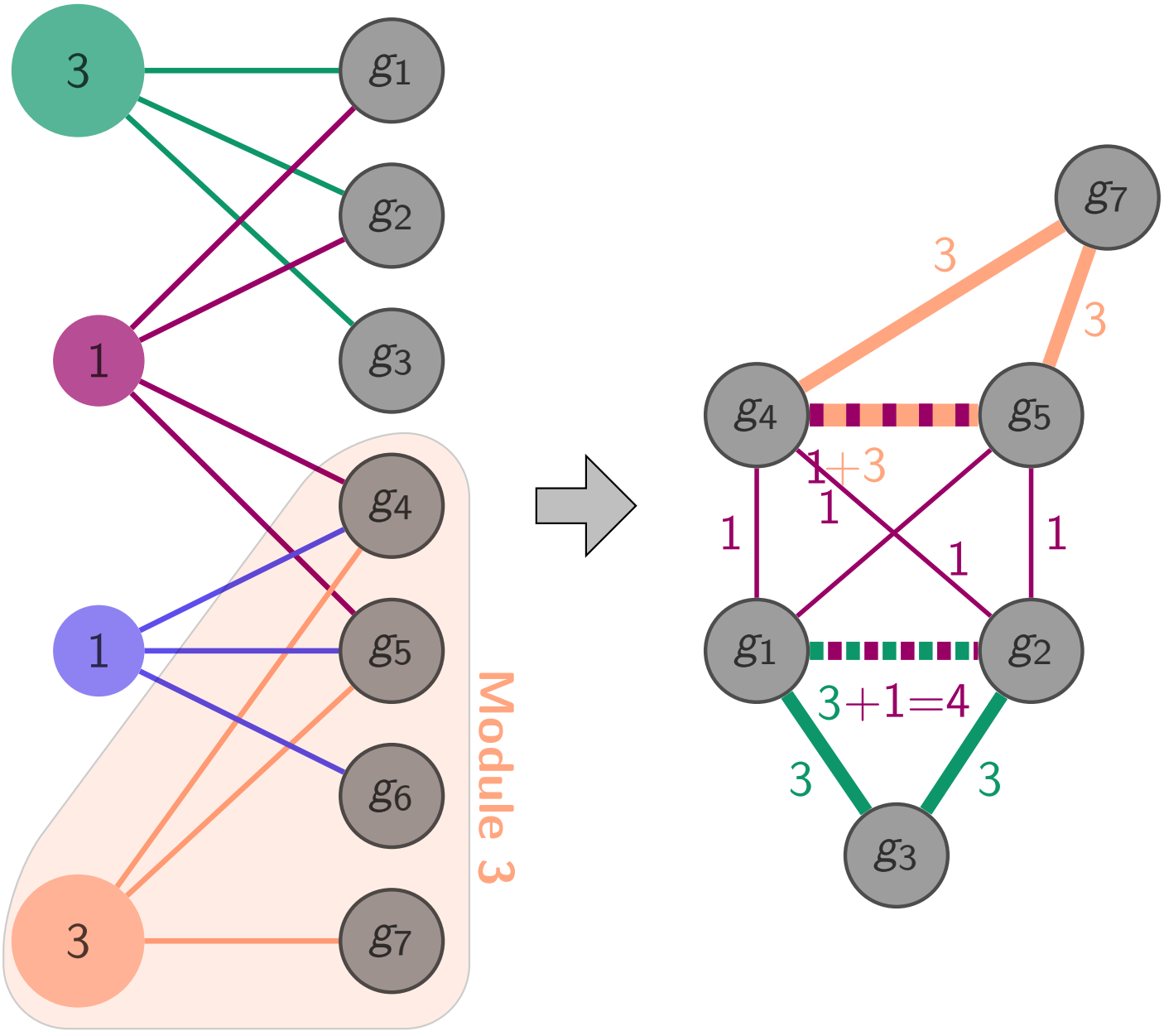
Gene-Gene Projection



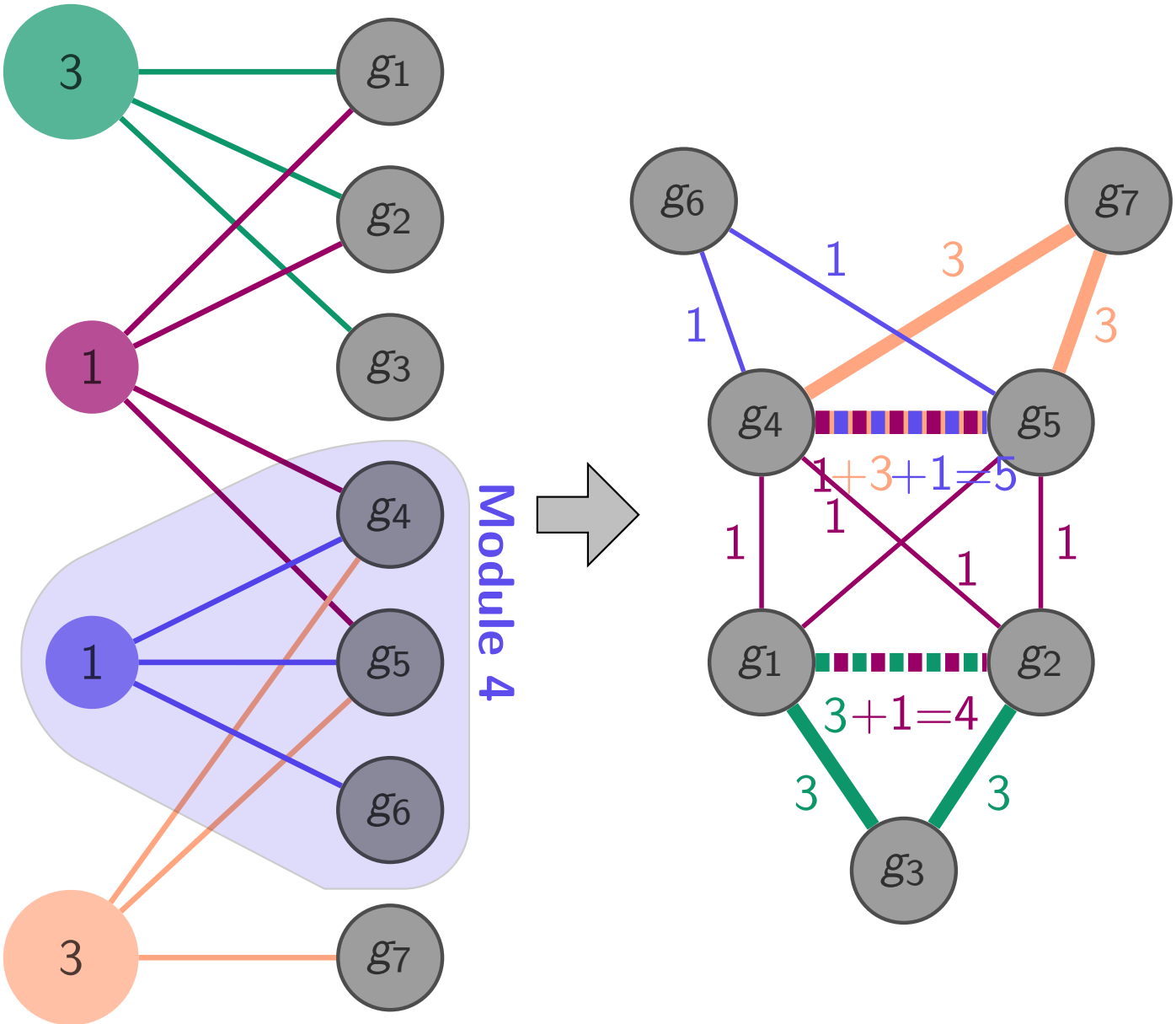
Gene-Gene Projection



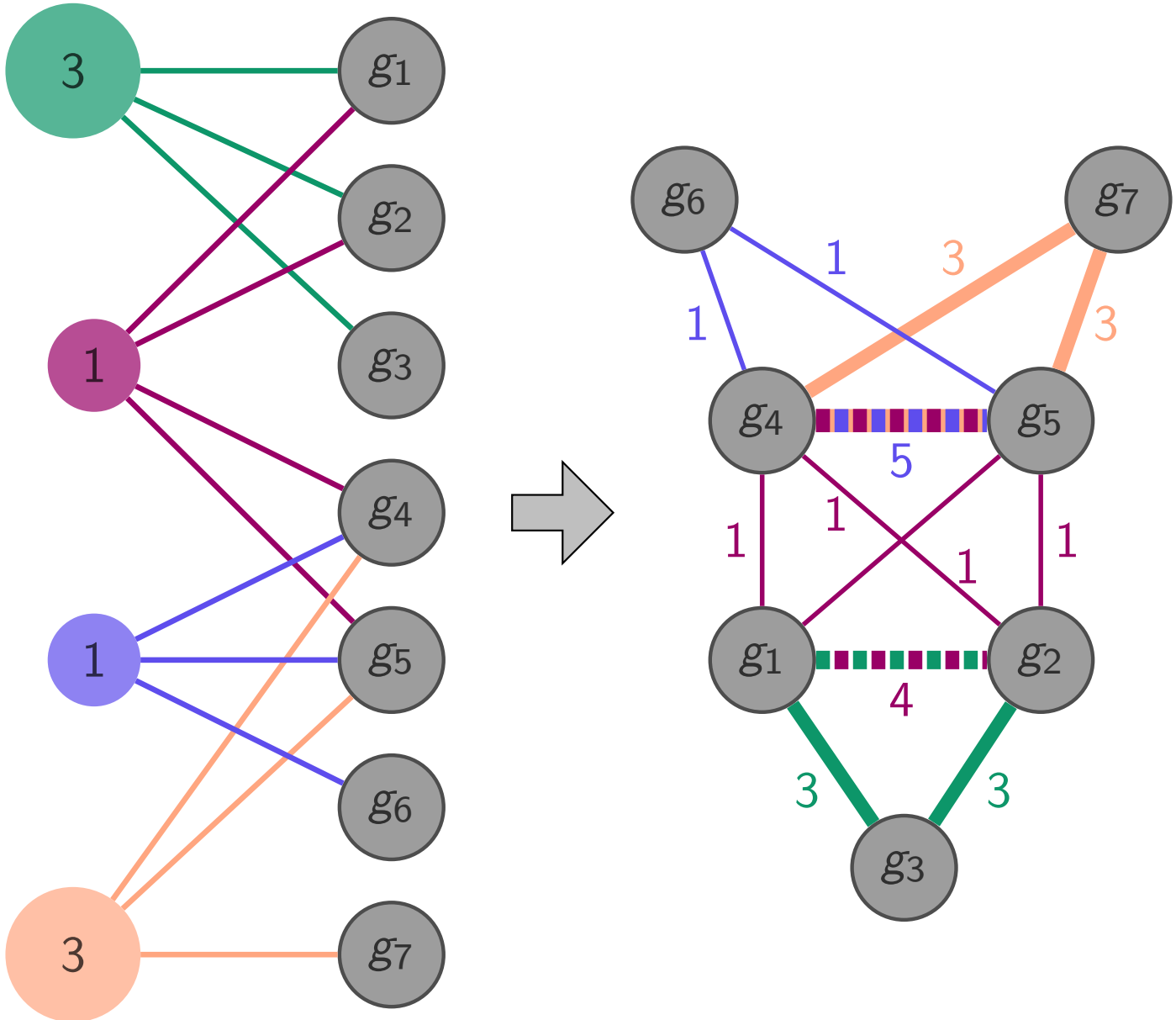
Gene-Gene Projection



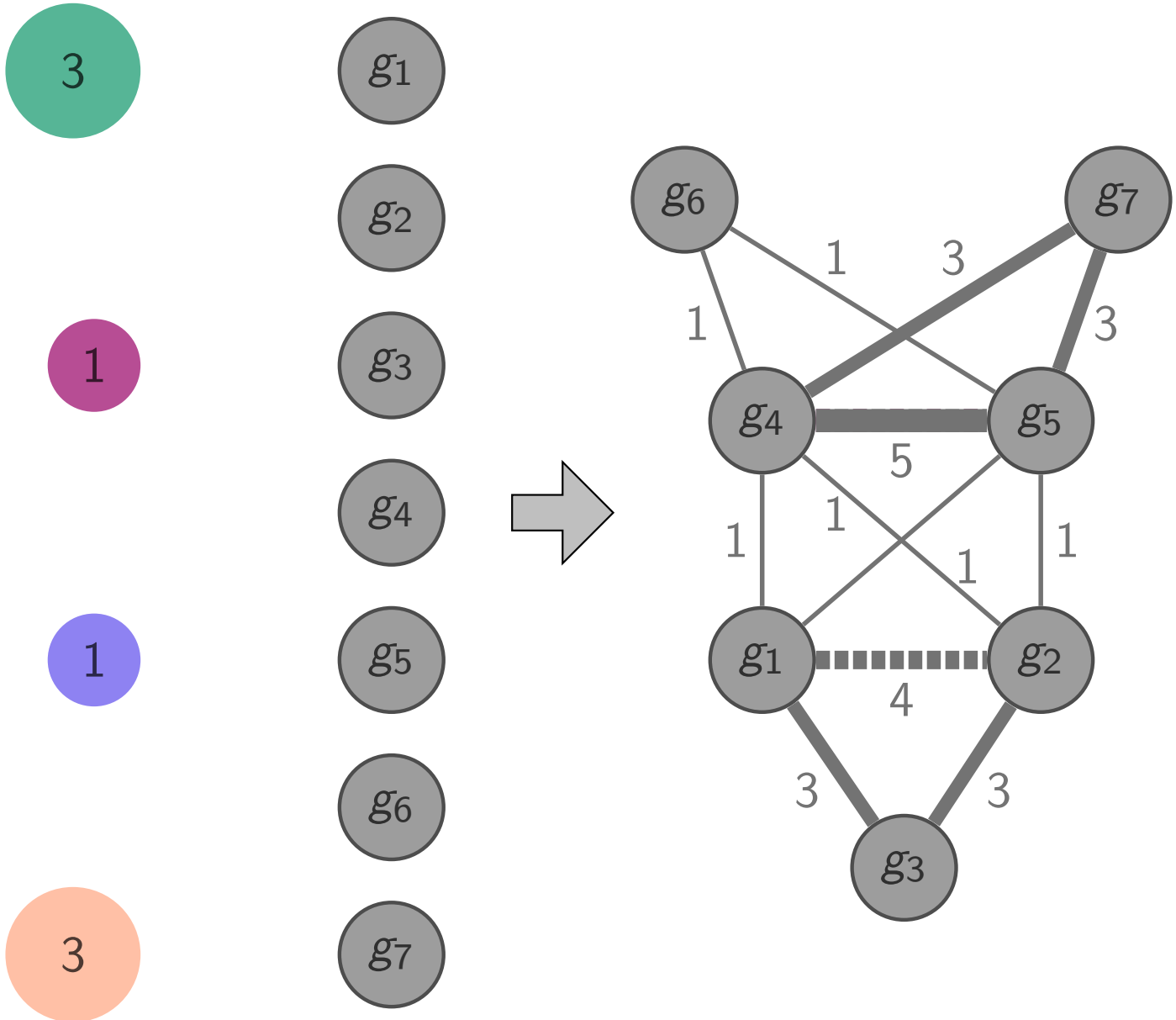
Gene-Gene Projection



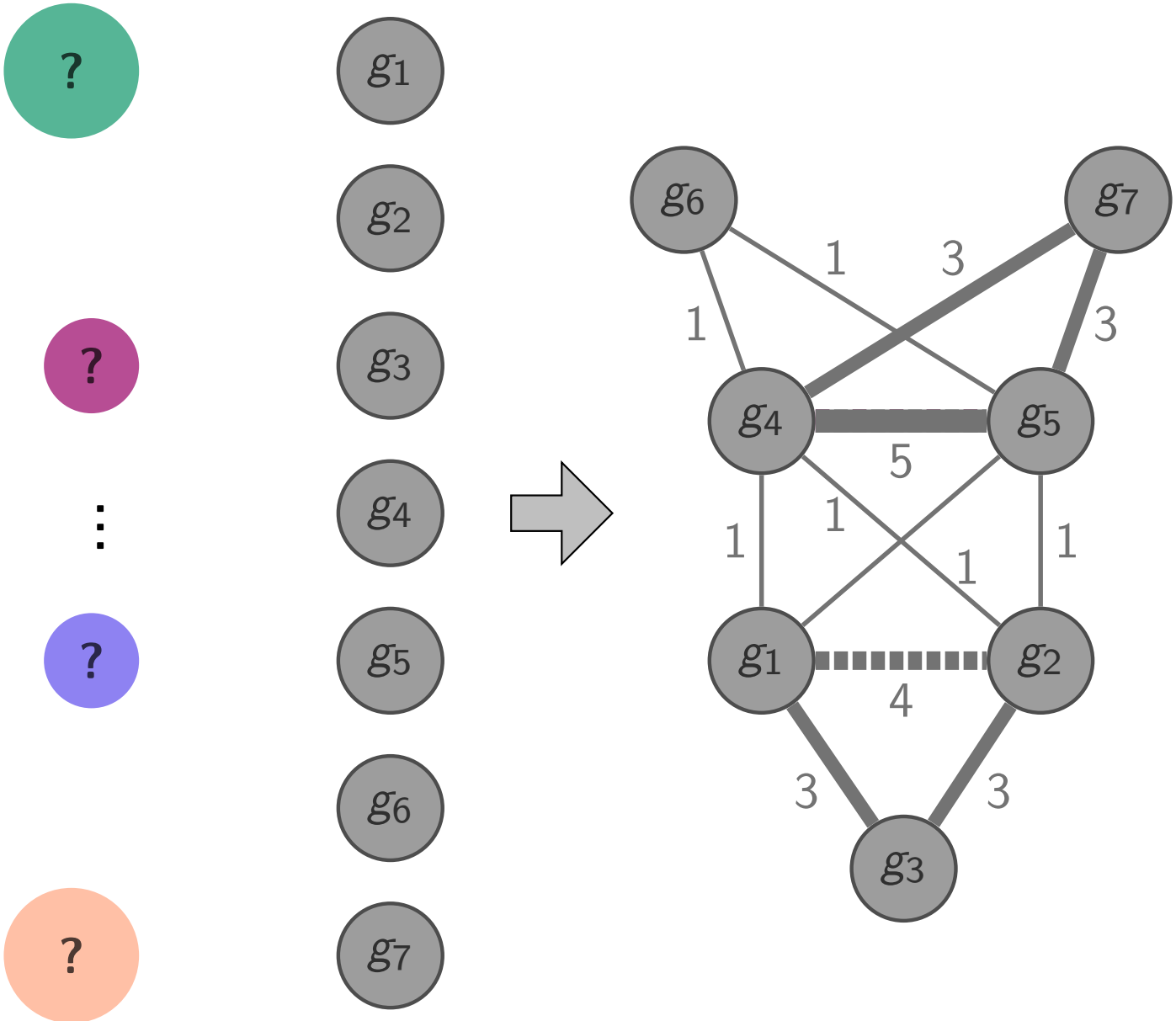
Challenges



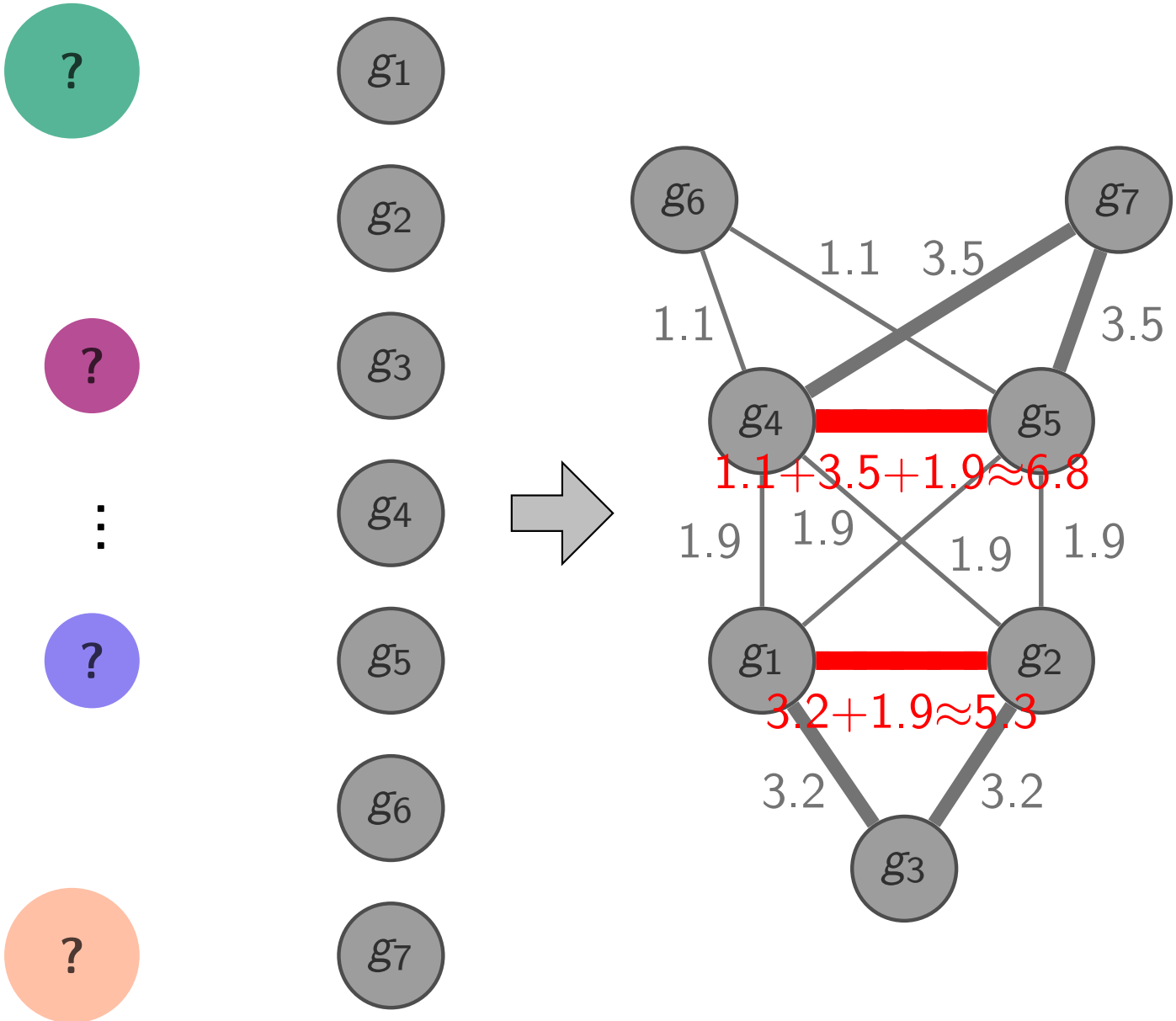
Challenges



Challenges

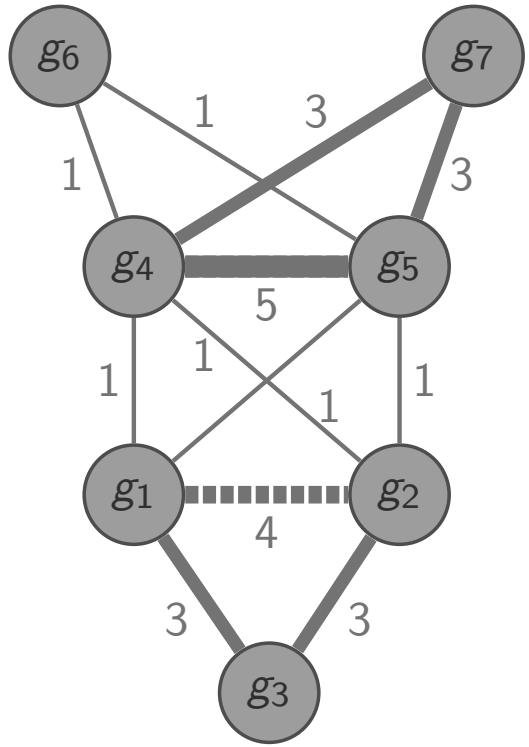


Challenges


?
?
?
?
g_1
g_2
g_3
g_4
g_5
g_6
g_7
g_6
g_7
1.1
3.5
1.1
3.5
g_4
g_5
1.1+3.5+1.9≈6.8
1.9
1.9
1.9
1.9
g_1
g_2
3.2+1.9≈5.3
3.2
3.2
g_3

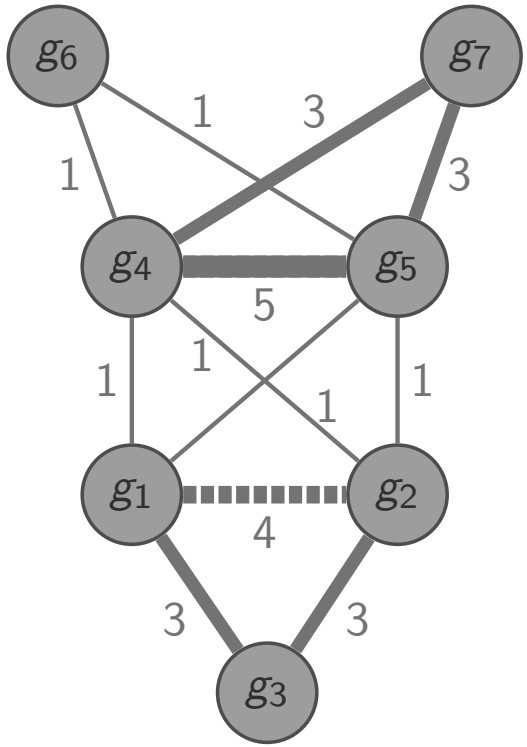
Restricted Setting

- Integer edge weights
- Exact edge sums



Restricted Setting

- Integer edge weights
- Exact edge sums



Exact Weighted Clique Decomposition (EWCD)

Input: a graph G, non-negative edge weights w, integer k.

Output: a set of at most k weighted cliques such that w agrees with the sum of containing cliques on each edge.

Approach

FPT Algorithms

Input: Input instance and additional parameter k.
Output: Solves problem exactly with runtime $f(k) \cdot |n|^{O(1)}$.

Approach

FPT Algorithms

Input: Input instance and additional parameter k.
Output: Solves problem exactly with runtime $f(k) \cdot |n|^{O(1)}$.

- Such a problem is fixed parameter tractable (FPT) with respect to parameter k

Approach

FPT Algorithms

Input: Input instance and additional parameter k.
Output: Solves problem exactly with runtime $f(k) \cdot |n|^{O(1)}$.

- Such a problem is fixed parameter tractable (FPT) with respect to parameter k
- Accompanied by kernelization algorithm that preprocesses to a smaller instance with bounded size

Approach

FPT Algorithms

Input: Input instance and additional parameter k.
Output: Solves problem exactly with runtime $f(k) \cdot |n|^{O(1)}$.

- Such a problem is fixed parameter tractable (FPT) with respect to parameter k
- Accompanied by kernelization algorithm that preprocesses to a smaller instance with bounded size
- Gives tractable algorithms for small k

Prior Work

- Feldmann et al. (2020) give $2^{O(K^{3/2}w^{1/2}\log(K/w))} + O(n^2 \log n)$ FPT algorithm for similar problem[2].

Weighted Edge Clique Partition

Input: a graph G, non-negative edge weights w, integer K.

Output: a set of at most K weight-1 cliques such that each edge appears in exactly as many cliques as w.

[2] with max-edge weight w

Prior Work

- Feldmann et al. (2020) give $2^{O(K^{3/2}w^{1/2}\log(K/w))} + O(n^2 \log n)$ FPT algorithm for similar problem[2].

Weighted Edge Clique Partition

Input: a graph G, non-negative edge weights w, integer K.

Output: a set of at most K weight-1 cliques such that each edge appears in exactly as many cliques as w.

- Decomposes into K *weight-1* cliques.

[2] with max-edge weight w

Prior Work

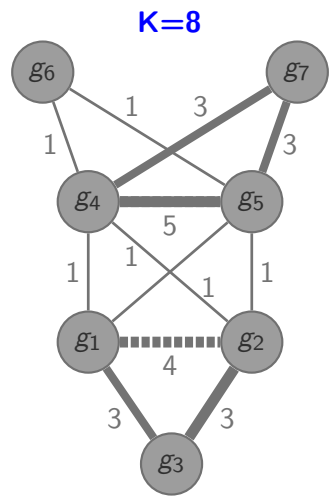
- Feldmann et al. (2020) give $2^{O(K^{3/2}w^{1/2}\log(K/w))} + O(n^2 \log n)$ FPT algorithm for similar problem[2].

Weighted Edge Clique Partition

Input: a graph G, non-negative edge weights w, integer K.

Output: a set of at most K weight-1 cliques such that each edge appears in exactly as many cliques as w.

- Decomposes into K *weight-1* cliques.



[2] with max-edge weight w

Prior Work

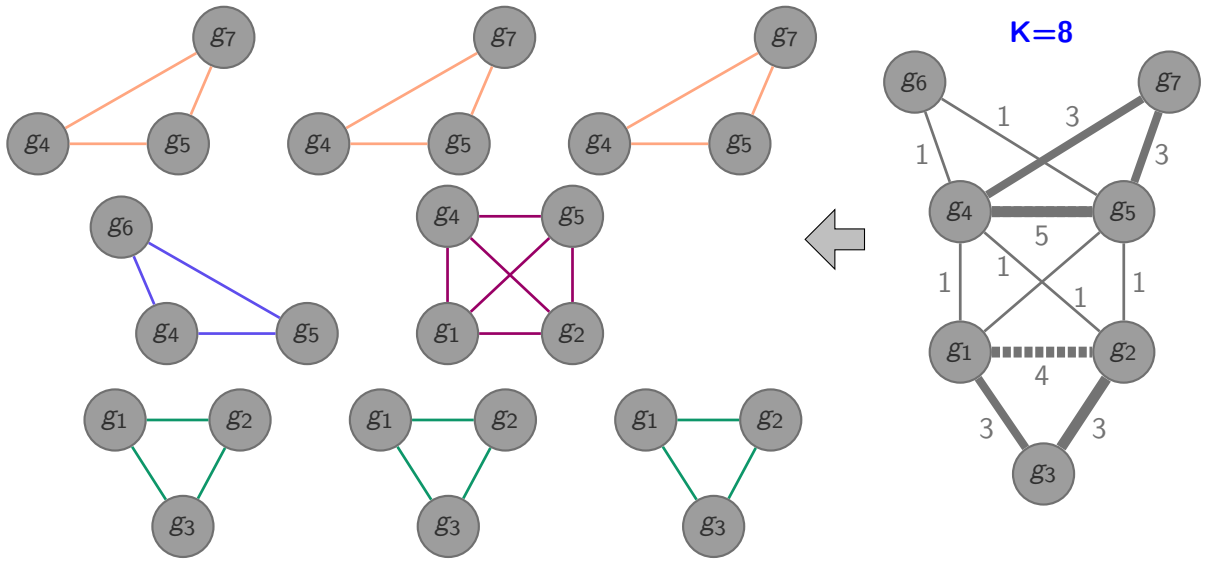
- Feldmann et al. (2020) give $2^{O(K^{3/2}w^{1/2}\log(K/w))} + O(n^2 \log n)$ FPT algorithm for similar problem[2].

Weighted Edge Clique Partition

Input: a graph G, non-negative edge weights w, integer K.

Output: a set of at most K weight-1 cliques such that each edge appears in exactly as many cliques as w.

- Decomposes into K *weight-1* cliques.



[2] with max-edge weight w

Prior Work

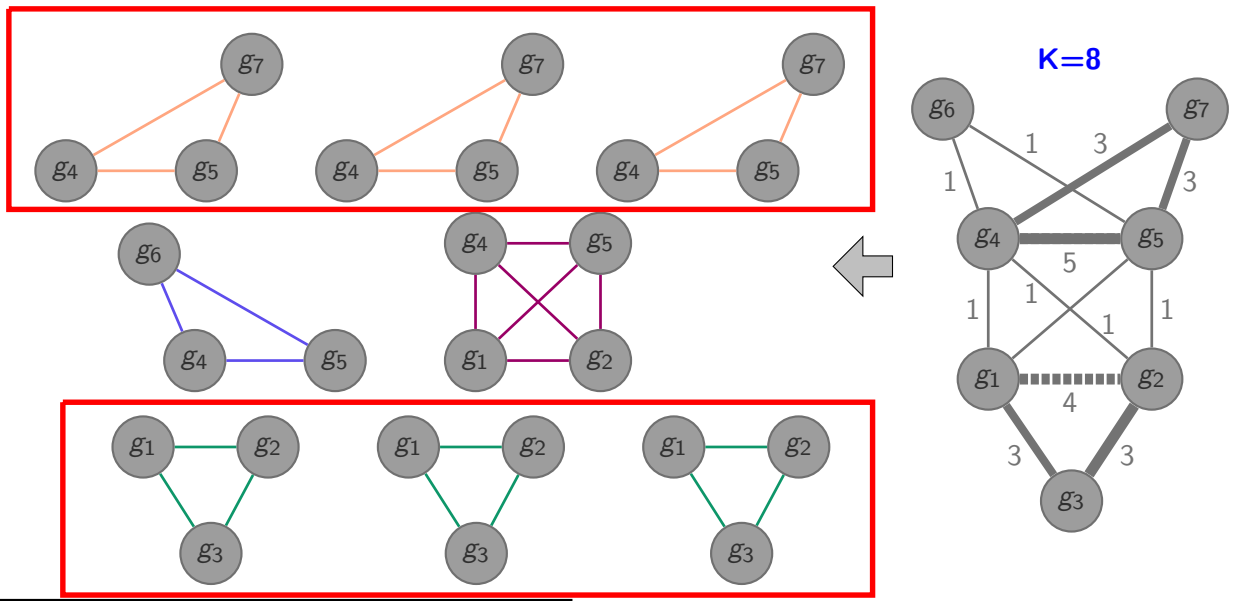
- Feldmann et al. (2020) give $2^{O(K^{3/2}w^{1/2}\log(K/w))} + O(n^2 \log n)$ FPT algorithm for similar problem[2].

Weighted Edge Clique Partition

Input: a graph G, non-negative edge weights w, integer K.

Output: a set of at most K weight-1 cliques such that each edge appears in exactly as many cliques as w.

- Decomposes into K *weight-1* cliques.



[2] with max-edge weight w

Contributions

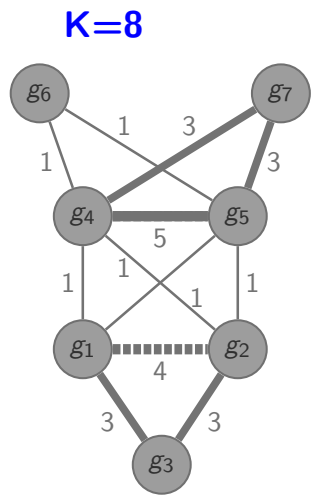
- Exact Weighted Clique Decomposition

Contributions

- Exact Weighted Clique Decomposition
- K weight-1 cliques $\rightarrow$ k weighted cliques
- $K \equiv$ *total # of cliques* $\rightarrow$ $k \equiv$ *# distinct cliques*

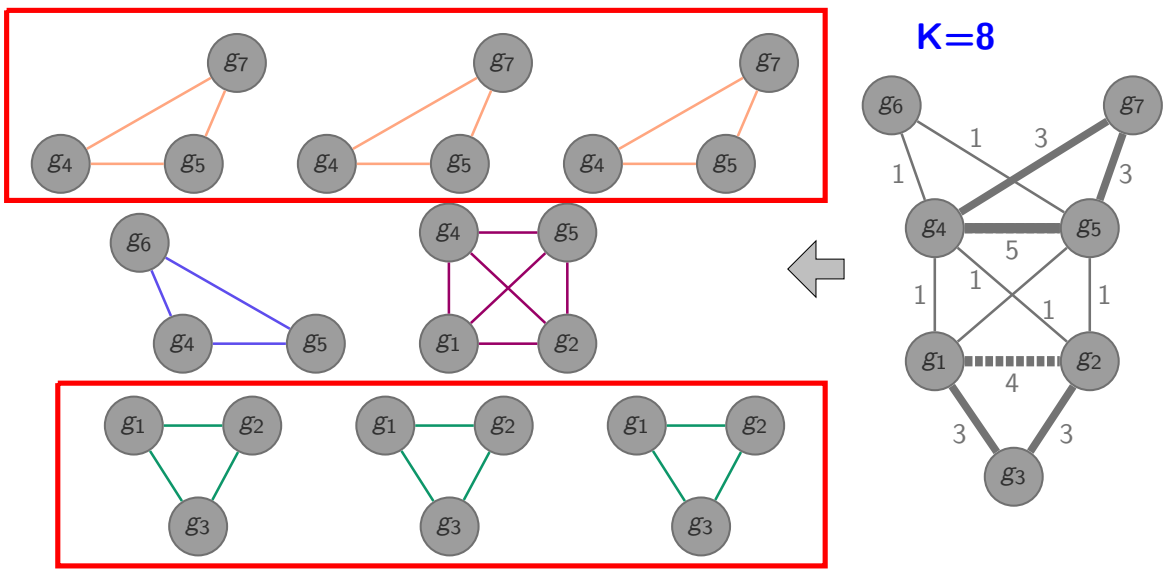
Contributions

- Exact Weighted Clique Decomposition
- K weight-1 cliques $\rightarrow$ k weighted cliques
- $K \equiv$ *total # of cliques* $\rightarrow$ $k \equiv$ *# distinct cliques*



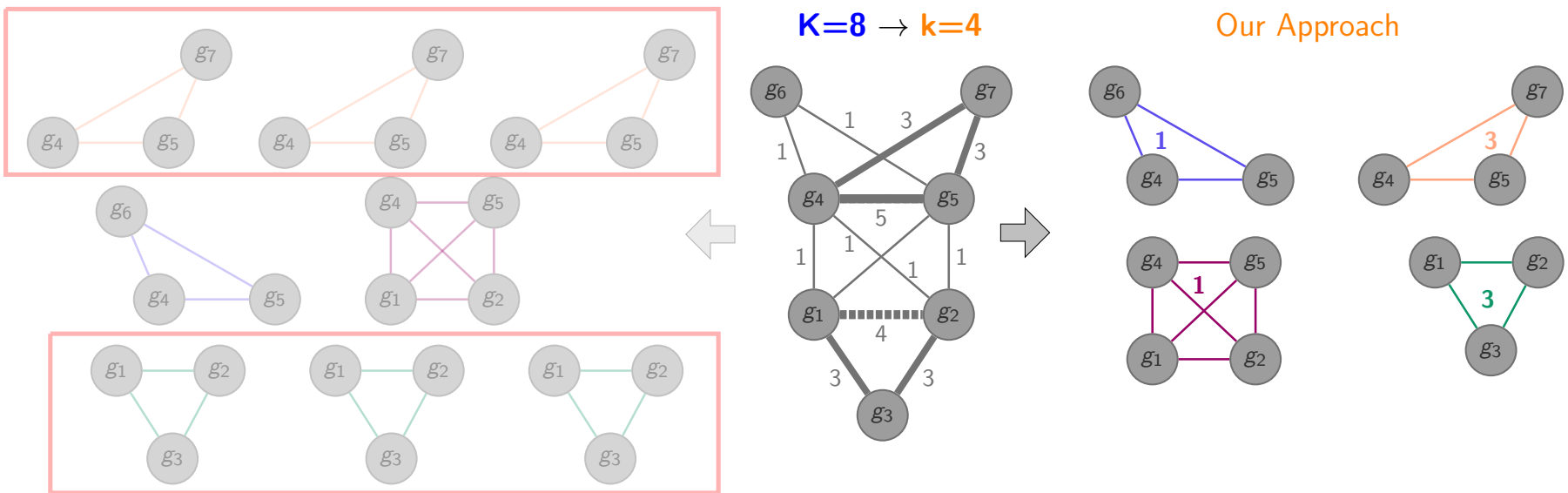
Contributions

- Exact Weighted Clique Decomposition
- K weight-1 cliques $\rightarrow$ k weighted cliques
- $K \equiv$ *total # of cliques* $\rightarrow$ $k \equiv$ *# distinct cliques*



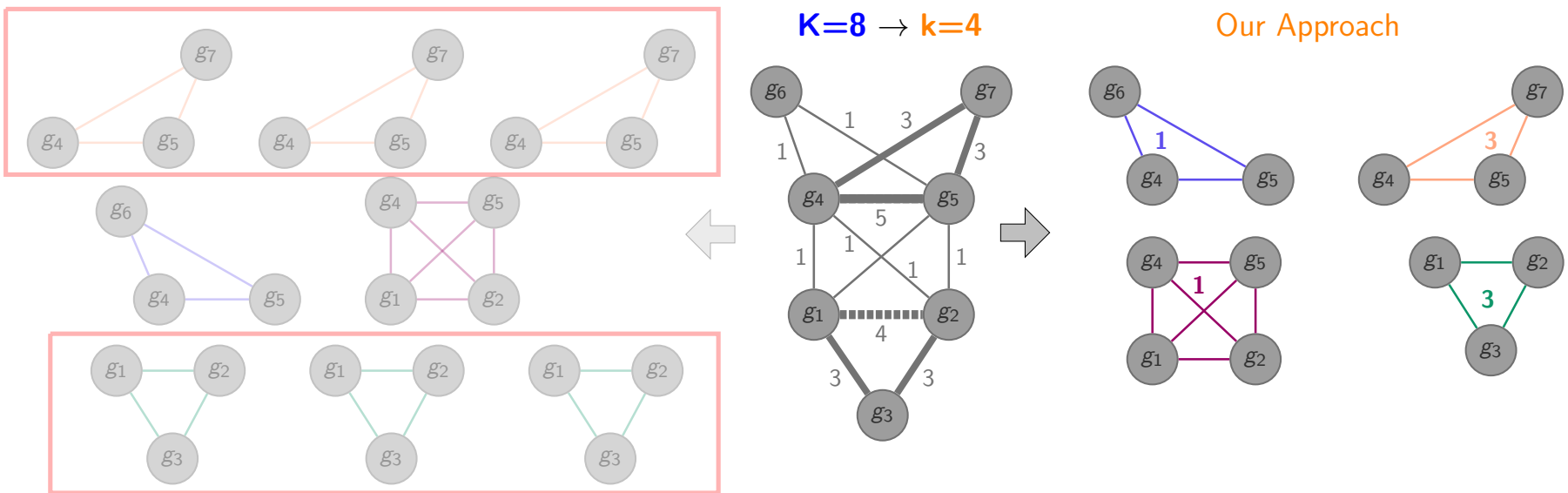
Contributions

- Exact Weighted Clique Decomposition
- *K* weight-1 cliques → *k* weighted cliques
- *K* ≡*total # of cliques* → *k* ≡ *# distinct cliques*



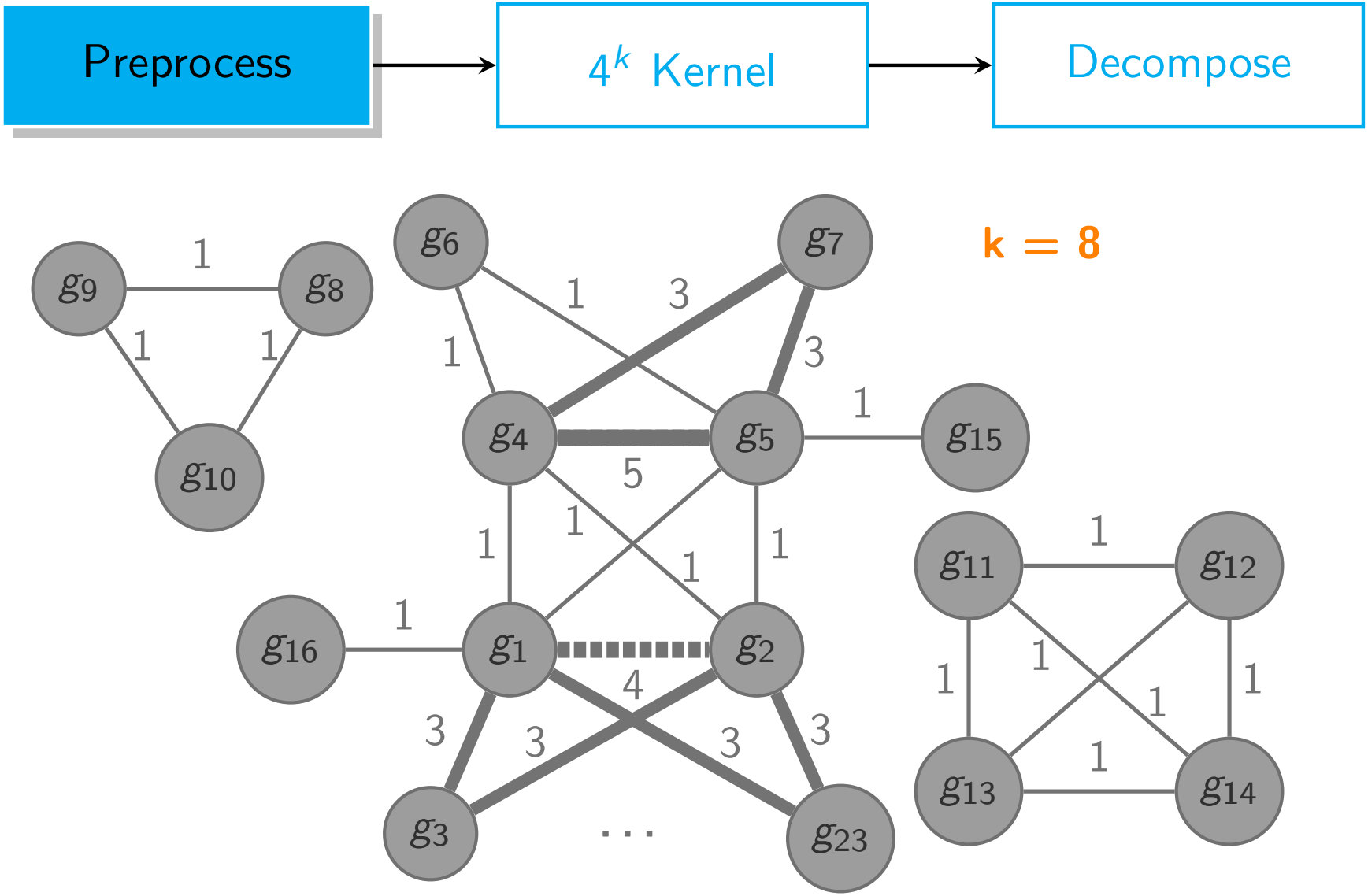
Contributions

- Exact Weighted Clique Decomposition
- K weight-1 cliques $\rightarrow$ k weighted cliques
- $K \equiv$ *total # of cliques* $\rightarrow$ $k \equiv$ *# distinct cliques*

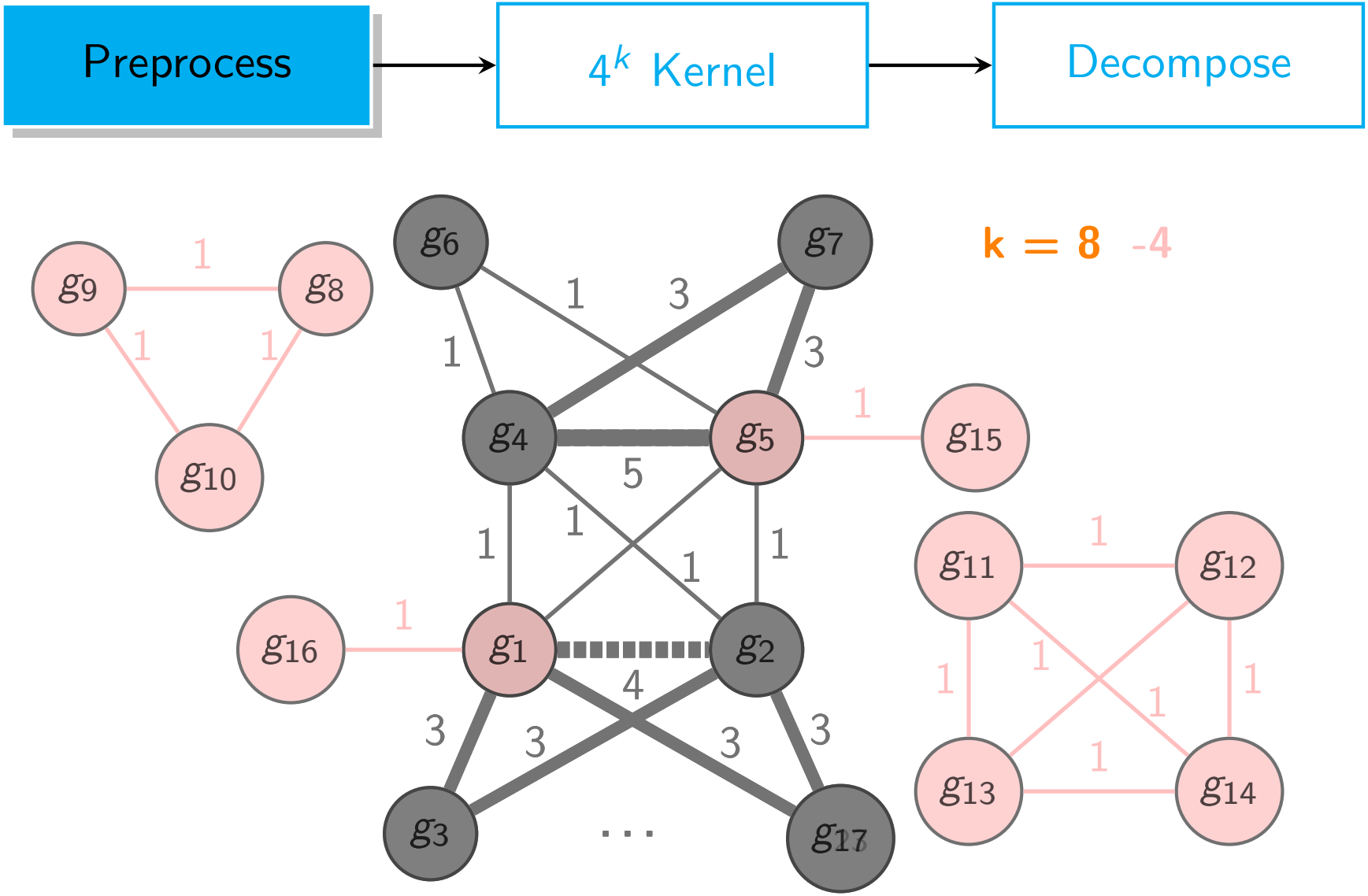


- Two FPT algorithms for solving EWCD & demonstrate improved scalability.

Pipeline

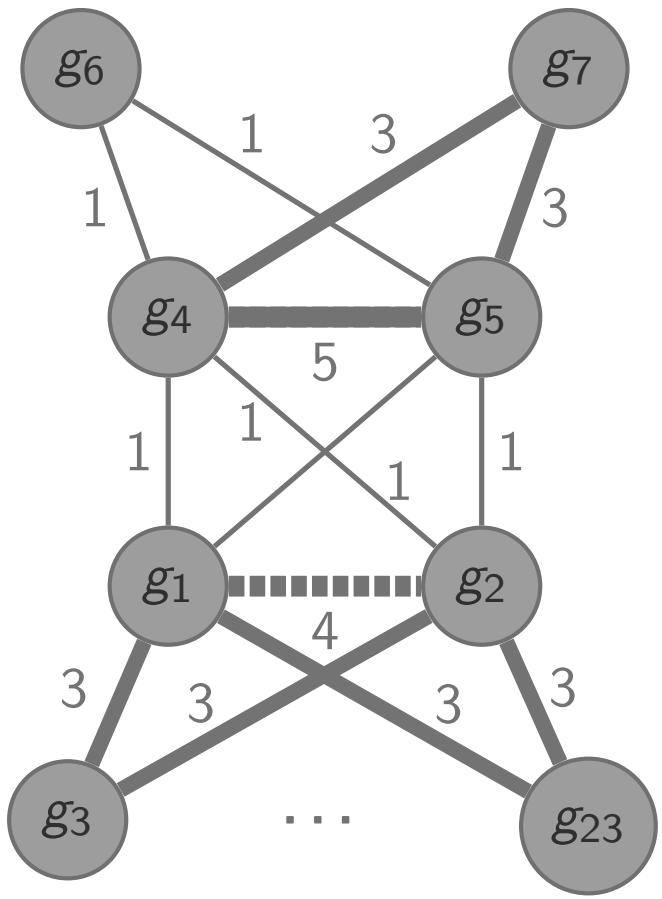

Preprocess
4^k Kernel
Decompose
k = 8
g_9
g_8
g_{10}
g_6
g_7
g_4
g_5
g_{15}
g_{16}
g_1
g_2
g_3
g_{23}
g_{11}
g_{12}
g_{13}
g_{14}

Pipeline


Preprocess
4^k Kernel
Decompose
k = 8 -4
g_6
g_7
g_9
g_8
g_4
g_5
g_{15}
g_{10}
g_{11}
g_{12}
g_{16}
g_1
g_2
g_{13}
g_{14}
g_3
g_{17}
...

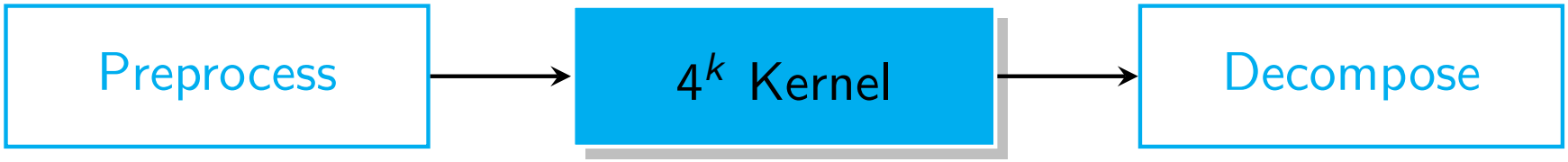
Pipeline

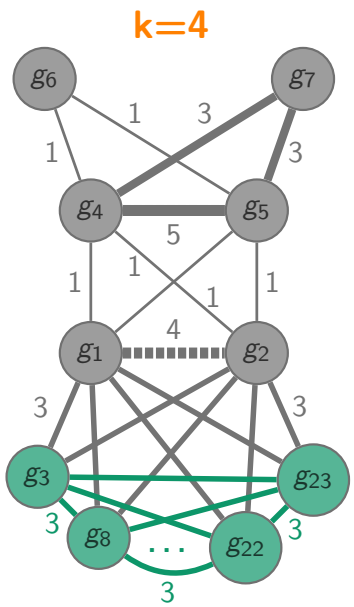

Preprocess
4^k Kernel
Decompose


g_6
g_7
g_4
g_5
g_1
g_2
g_3
g_{23}
1
3
1
3
5
1
1
1
1
4
3
3
3
3
...

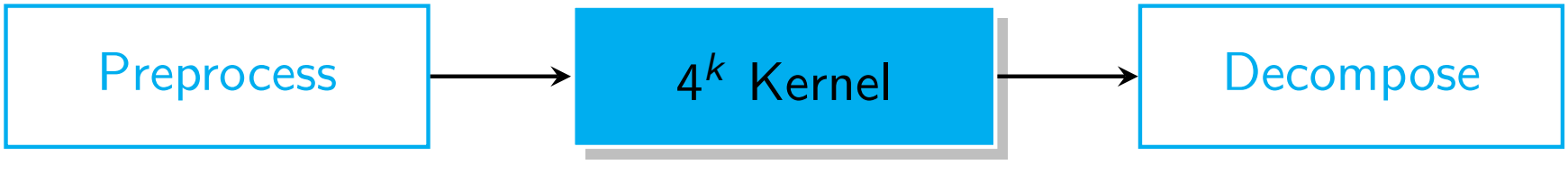
k = 4

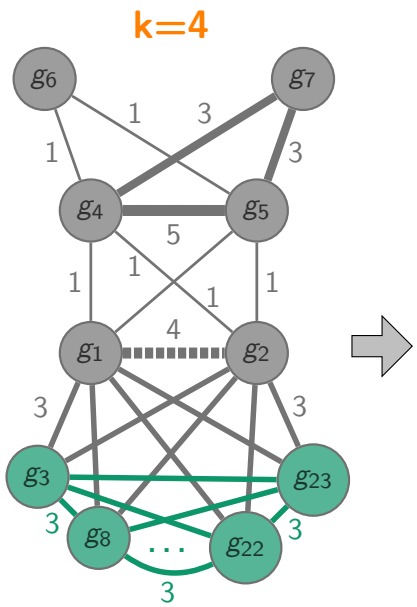
Pipeline


Preprocess
4^k Kernel
Decompose


k=4
g_6
g_7
g_4
g_5
g_1
g_2
g_3
g_{23}
g_8
g_{22}
...
1
3
5
4

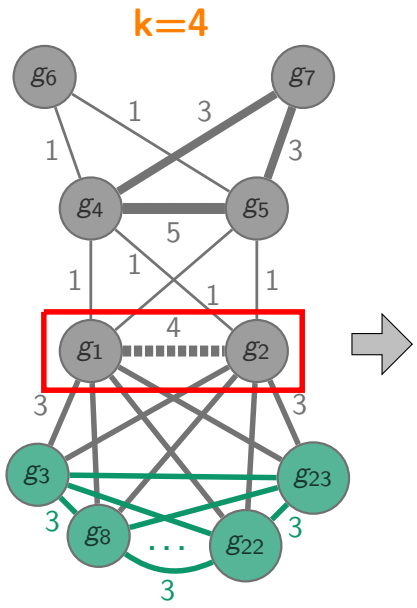
Pipeline





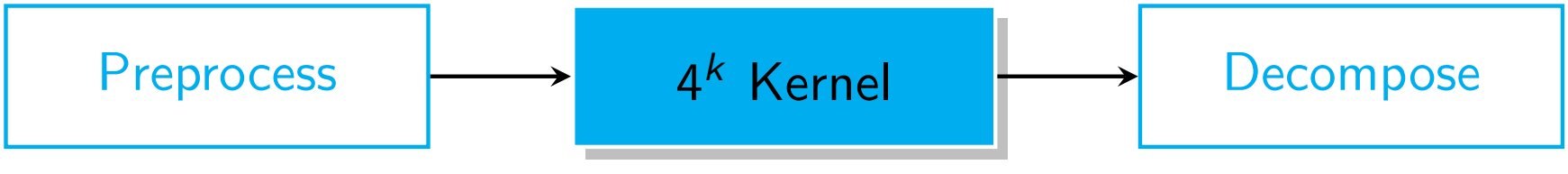
1. Combine *twin* vertices into *blocks*

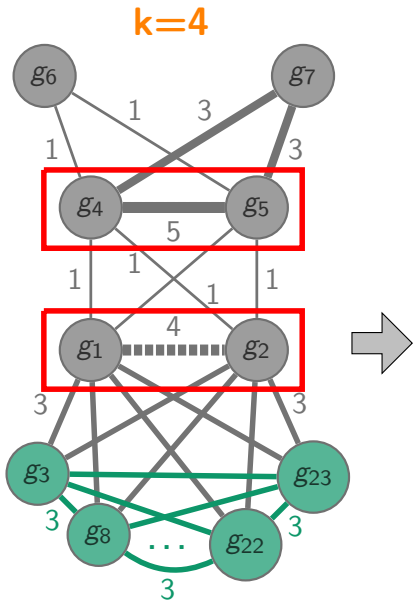
Pipeline



1. Combine *twin* vertices into *blocks*

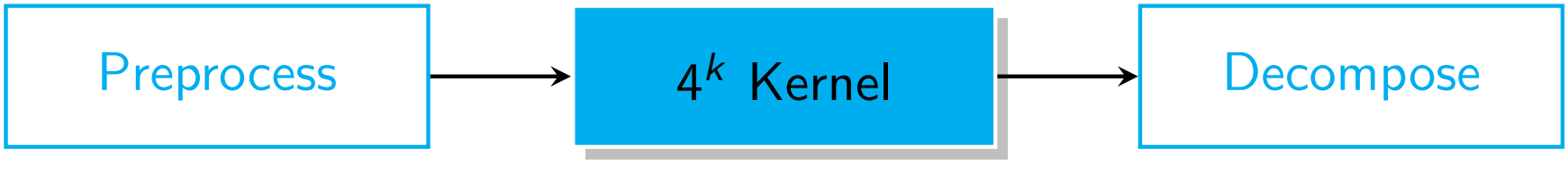
Pipeline

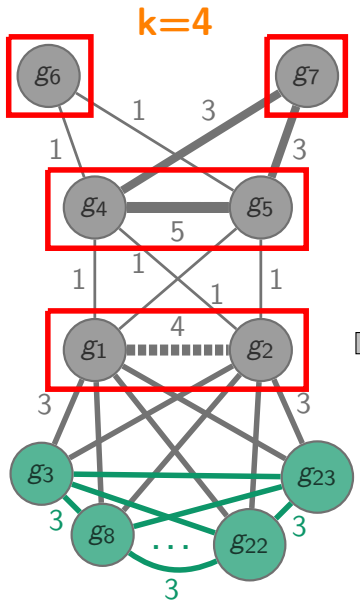




1. Combine *twin* vertices into *blocks*

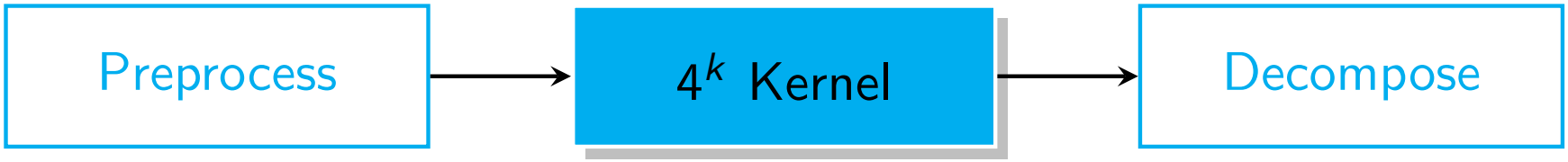
Pipeline

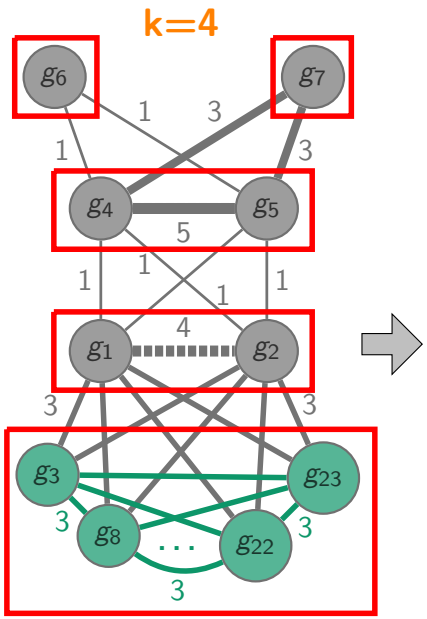




1. Combine *twin* vertices into *blocks*

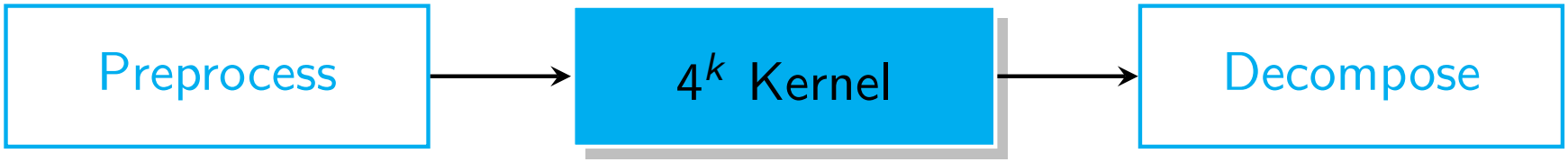
Pipeline

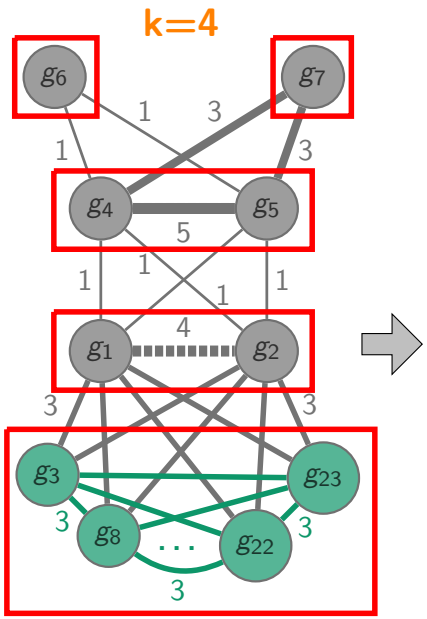




1. Combine *twin* vertices into *blocks*

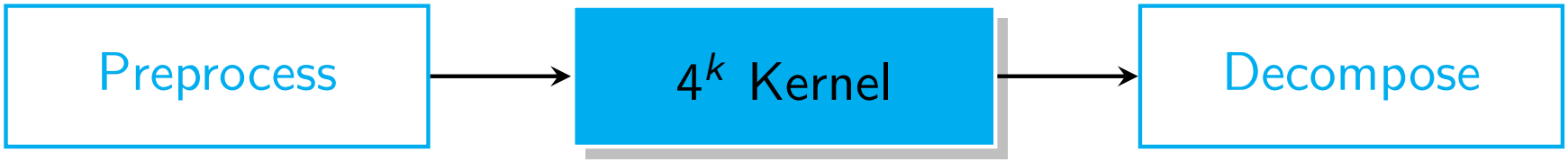
Pipeline

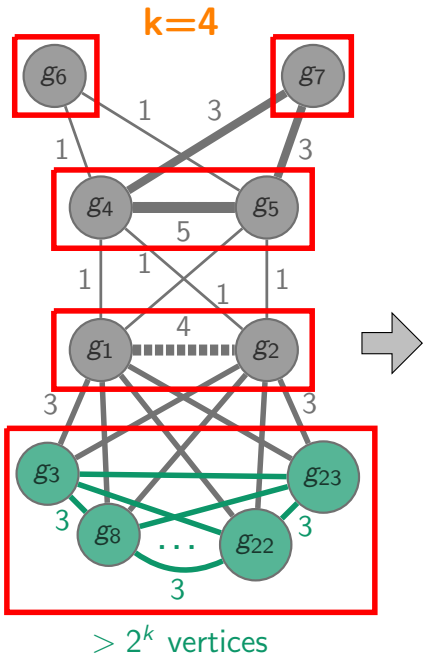




1. Combine *twin* vertices into *blocks*
2. If # blocks $> 2^k \rightarrow$ NO

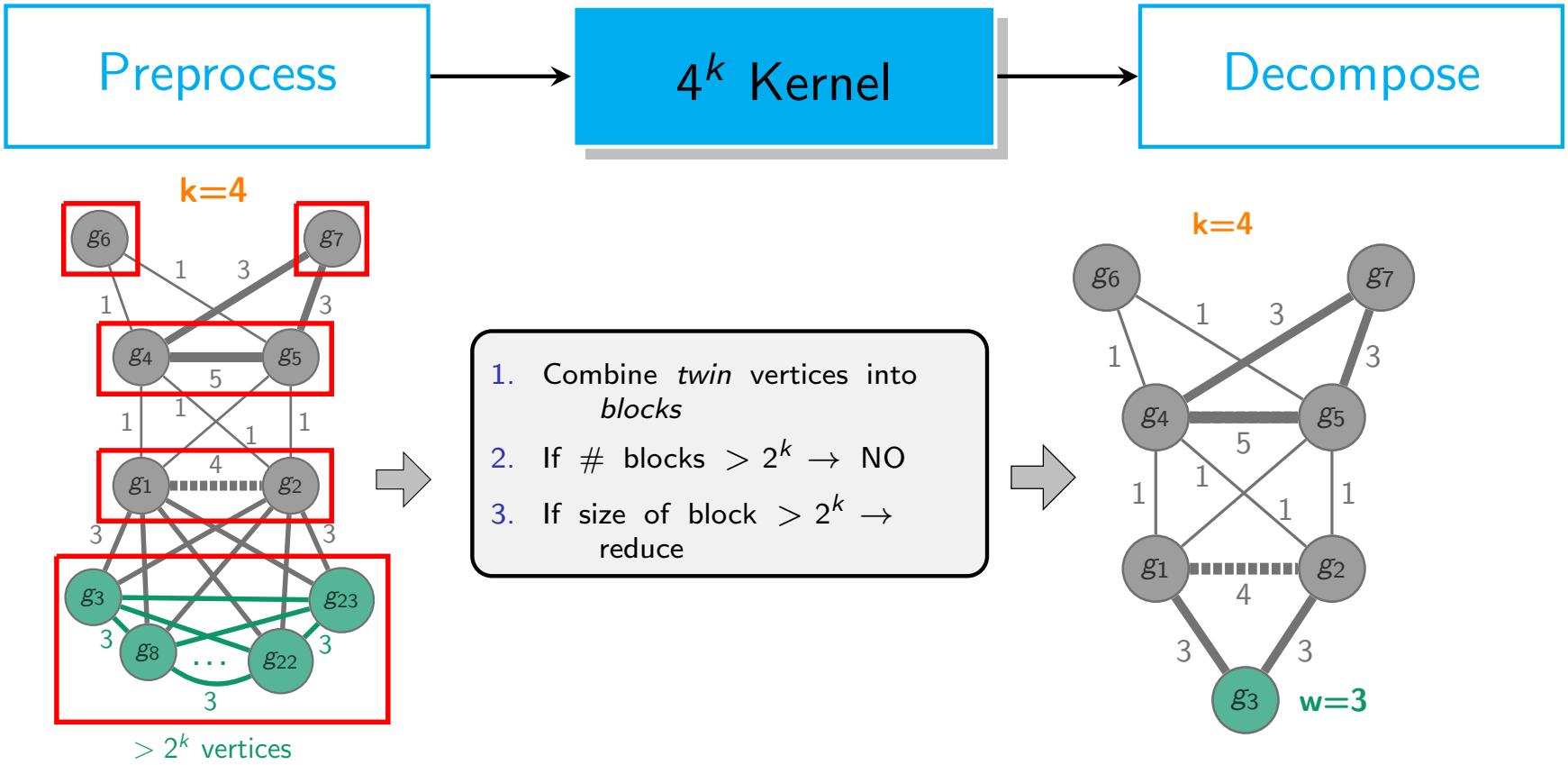
Pipeline



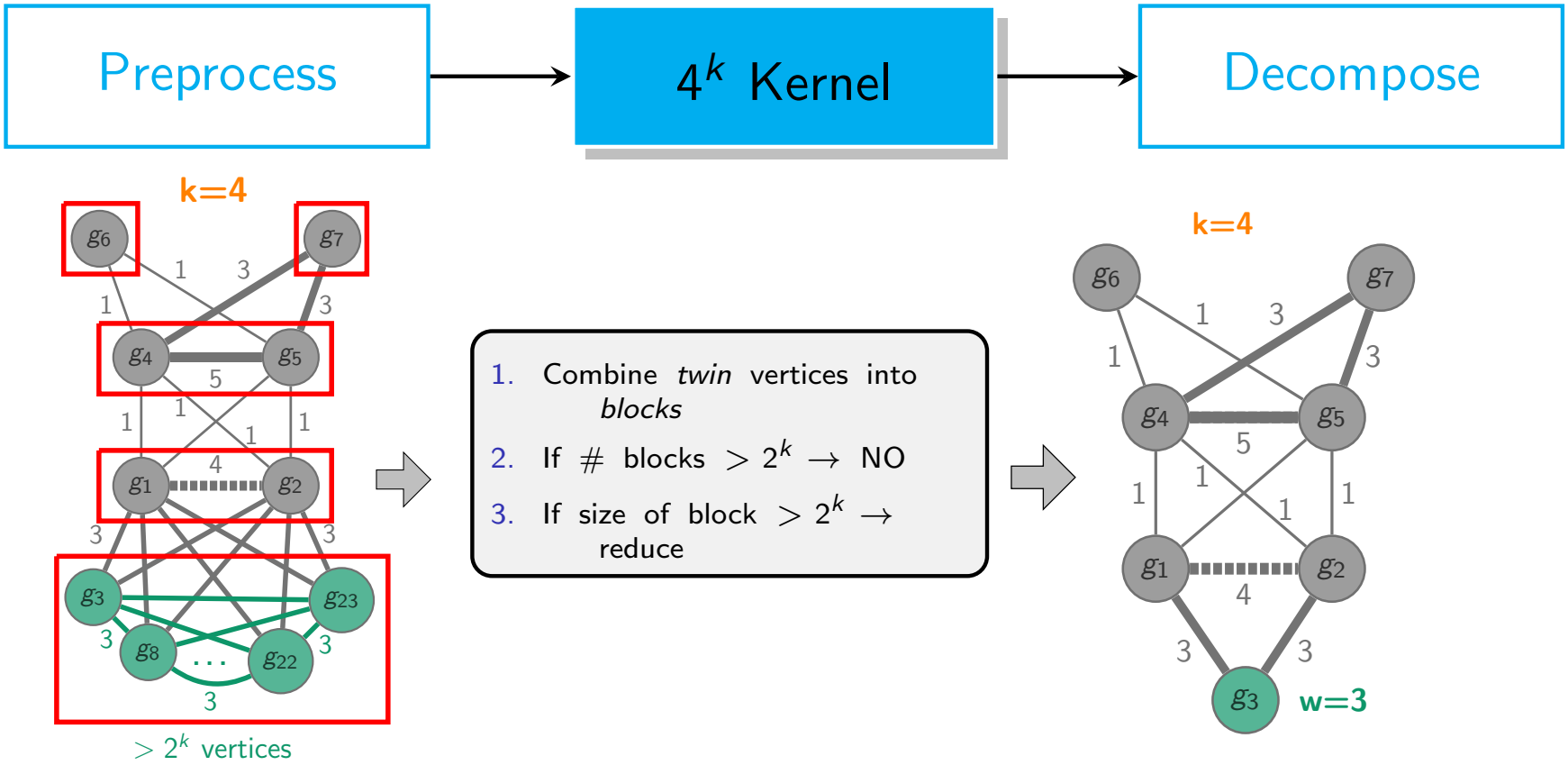


1. Combine *twin* vertices into *blocks*
2. If # blocks $> 2^k \rightarrow$ NO
3. If size of block $> 2^k \rightarrow$ reduce

Pipeline


Preprocess
4^k Kernel
Decompose
k=4
g6
g7
g4
g5
g1
g2
g3
g23
g8
g22
> 2^k vertices
1. Combine twin vertices into blocks
2. If # blocks > 2^k → NO
3. If size of block > 2^k → reduce
k=4
w=3

Pipeline



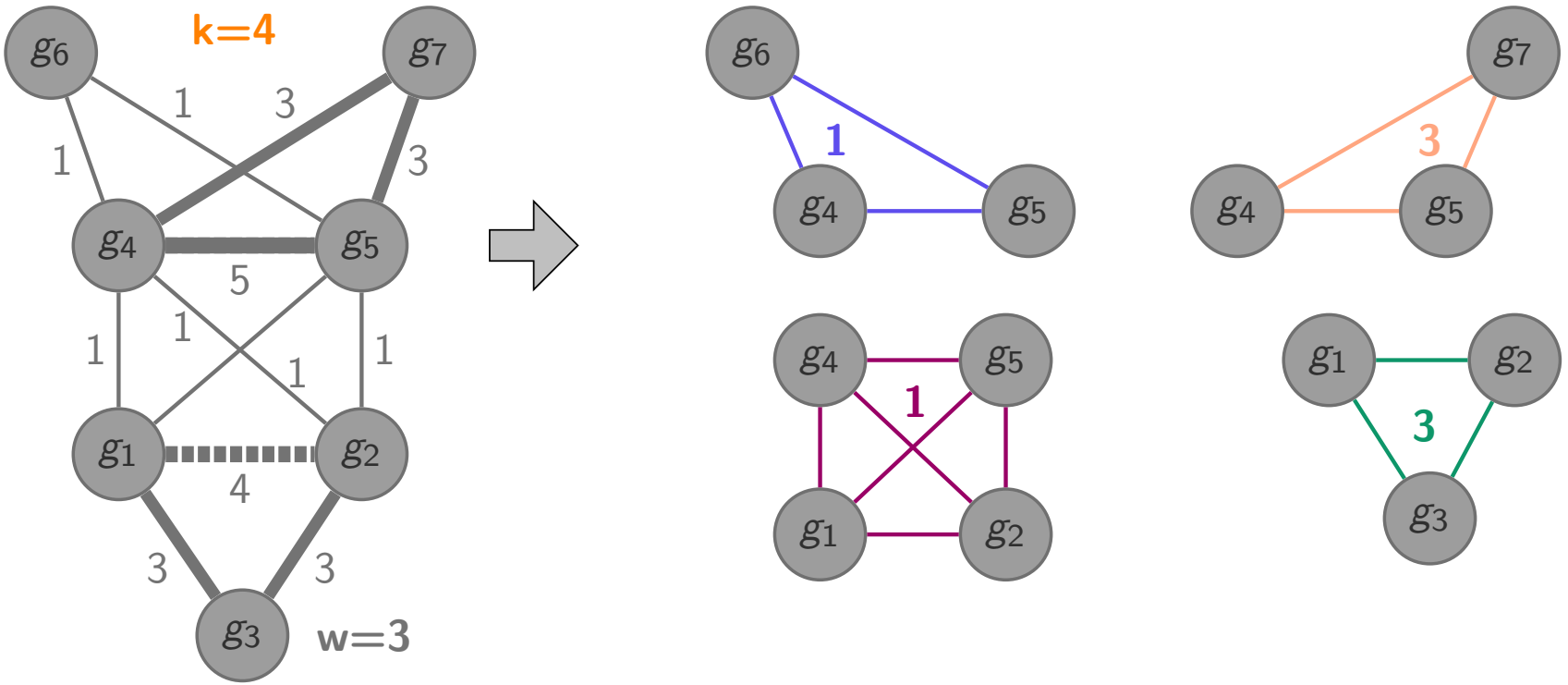
Annotated EWCD

Input: a graph G, non-negative edge weights, set of vertex weights, integer k.

Output: a set of at most k weighted cliques such that the sum of containing cliques on each edge agrees with the edge weight.

Pipeline


Preprocess
4^k Kernel
Decompose

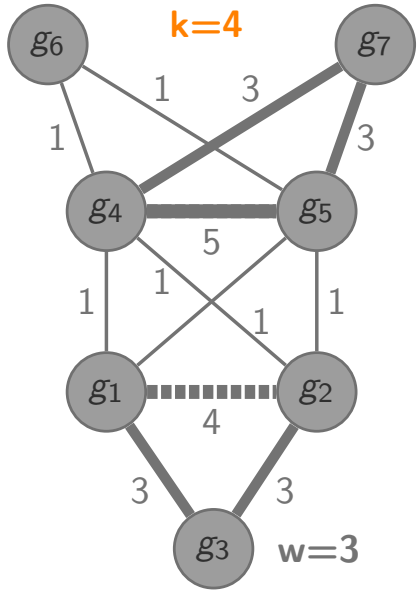

k=4
g_6
g_7
g_4
g_5
g_1
g_2
g_3
1
3
1
3
5
1
1
1
1
4
3
3
w=3
g_6
g_4
g_5
1
g_7
g_4
g_5
3
g_4
g_5
g_1
g_2
1
g_1
g_2
g_3
3

Matrix Reformulation

- Both algorithms solve equivalent matrix reformulations

Matrix Reformulation

- Both algorithms solve equivalent matrix reformulations

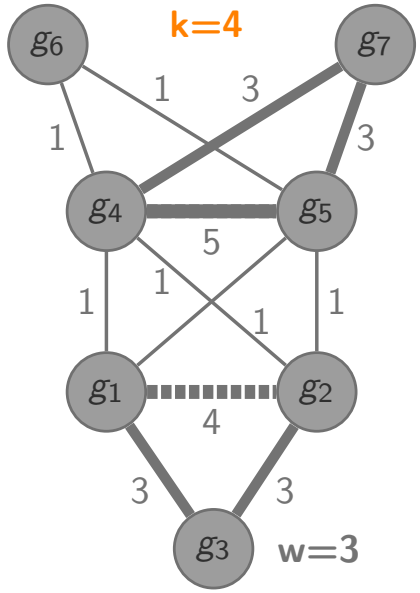


	g_1	g_2	g_3	g_4	g_5	g_6	g_7
g_1	$*$	4	3	1	1	0	0
g_2	4	$*$	3	1	1	0	0
g_3	3	3	3	0	0	0	0
g_4	1	1	0	$*$	5	1	3
g_5	1	1	0	5	$*$	1	3
g_6	0	0	0	1	1	$*$	0
g_7	0	0	0	3	3	0	$*$

$= A$

Matrix Reformulation

- Both algorithms solve equivalent matrix reformulations



	g_1	g_2	g_3	g_4	g_5	g_6	g_7
g_1	$*$	4	3	1	1	0	0
g_2	4	$*$	3	1	1	0	0
g_3	3	3	3	0	0	0	0
g_4	1	1	0	$*$	5	1	3
g_5	1	1	0	5	$*$	1	3
g_6	0	0	0	1	1	$*$	0
g_7	0	0	0	3	3	0	$*$

$= A$

BSWD-DW

Input: Symmetric matrix A with diagonal wildcards, integer k.

Output[a]*:* $n \times k$ binary matrix B, diagonal $k \times k$ matrix W s.t. $A \overset{\star}{=} BWB^T$.

[a] where $\overset{\star}{=}$ denotes that wildcards are equal to any number

Equivalent Problems

B

	C_1	C_2	C_3	C_4
g_1	1	1	0	0
g_2	1	1	0	0
g_3	1	0	0	0
g_4	0	1	1	1
g_5	0	1	1	1
g_6	0	0	1	0
g_7	0	0	0	1

W

C_1	C_2	C_3	C_4
3			
	1		
		1	
			3

B^T

g_1	g_2	g_3	g_4	g_5	g_6	g_7
1	1	1	0	0	0	0
1	1	0	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	1

$\stackrel{\star}{=} \quad A$

Equivalent Problems

B

	C_1	C_2	C_3	C_4
g_1	1	1	0	0
g_2	1	1	0	0
g_3	1	0	0	0
g_4	0	1	1	1
g_5	0	1	1	1
g_6	0	0	1	0
g_7	0	0	0	1

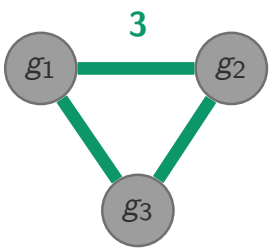
W

C_1	C_2	C_3	C_4
3			
	1		
		1	
			3

B^T

g_1	g_2	g_3	g_4	g_5	g_6	g_7
1	1	1	0	0	0	0
1	1	0	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	1

$\stackrel{\star}{=} A$



Equivalent Problems

B

	C_1	C_2	C_3	C_4
g_1	1	1	0	0
g_2	1	1	0	0
g_3	1	0	0	0
g_4	0	1	1	1
g_5	0	1	1	1
g_6	0	0	1	0
g_7	0	0	0	1

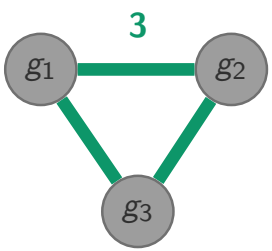
W

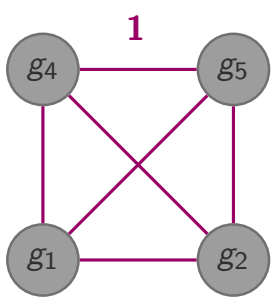
C_1	C_2	C_3	C_4
3			
	1		
		1	
			3

B^T

g_1	g_2	g_3	g_4	g_5	g_6	g_7
1	1	1	0	0	0	0
1	1	0	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	1

$\stackrel{\star}{=} A$





Equivalent Problems

B

	C_1	C_2	C_3	C_4
g_1	1	1	0	0
g_2	1	1	0	0
g_3	1	0	0	0
g_4	0	1	1	1
g_5	0	1	1	1
g_6	0	0	1	0
g_7	0	0	0	1

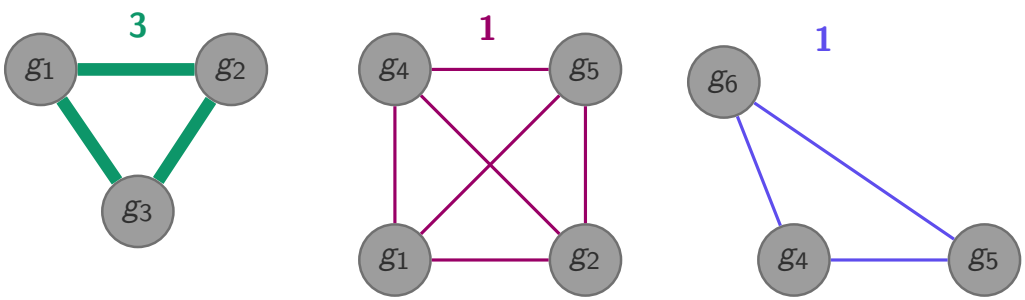
W

C_1	C_2	C_3	C_4
3			
	1		
		1	
			3

B^T

g_1	g_2	g_3	g_4	g_5	g_6	g_7
1	1	1	0	0	0	0
1	1	0	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	1

$\stackrel{\star}{=} A$



Equivalent Problems

B

	C_1	C_2	C_3	C_4
g_1	1	1	0	0
g_2	1	1	0	0
g_3	1	0	0	0
g_4	0	1	1	1
g_5	0	1	1	1
g_6	0	0	1	0
g_7	0	0	0	1

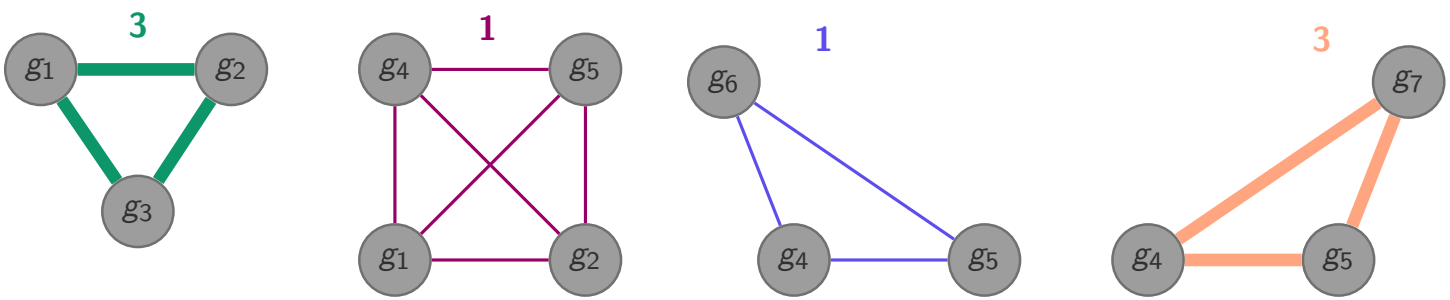
W

C_1	C_2	C_3	C_4
3			
	1		
		1	
			3

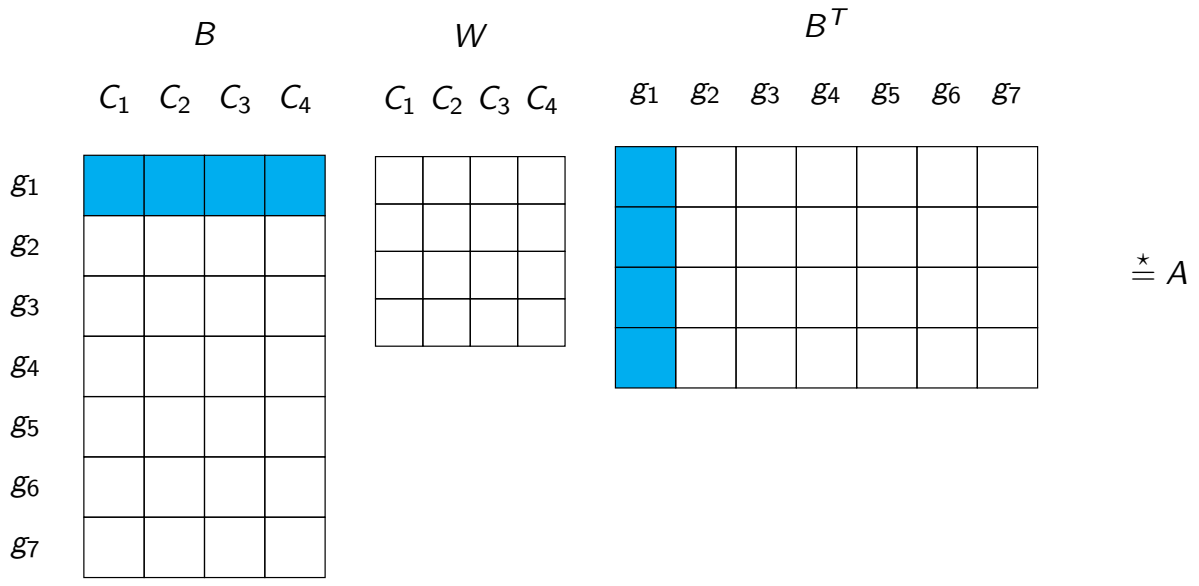
B^T

g_1	g_2	g_3	g_4	g_5	g_6	g_7
1	1	1	0	0	0	0
1	1	0	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	1

$\stackrel{\star}{=} \quad A$

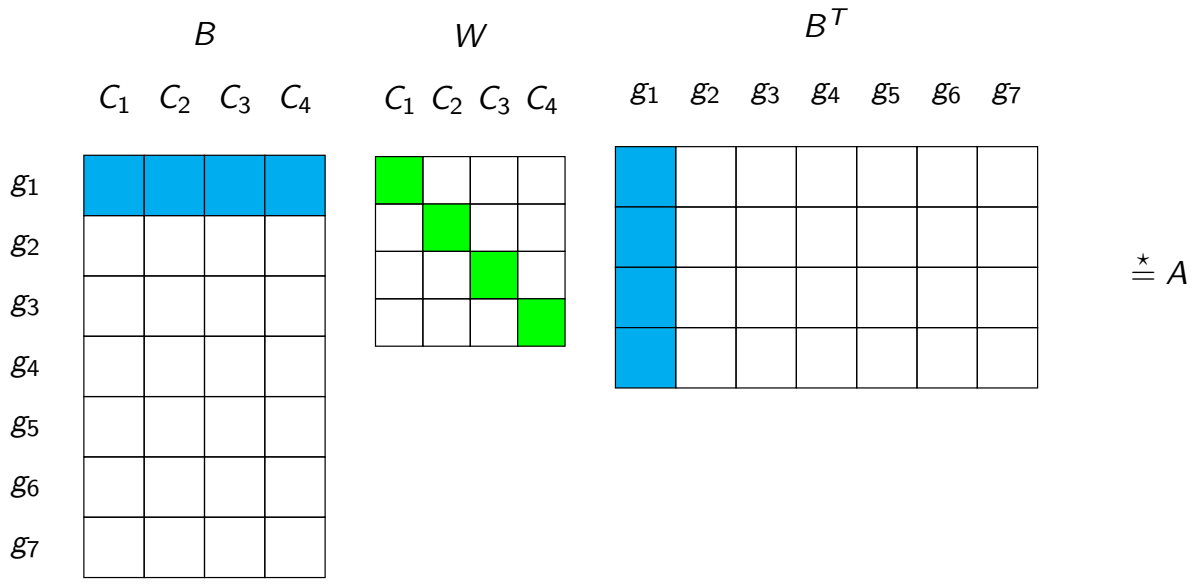


Algorithm Overview



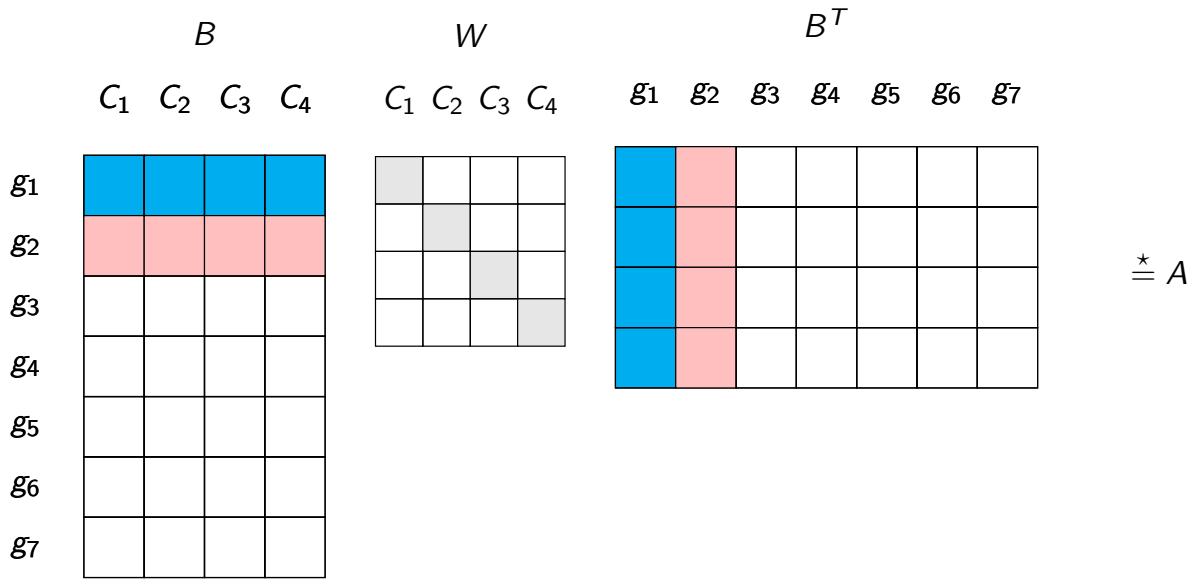
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Algorithm Overview



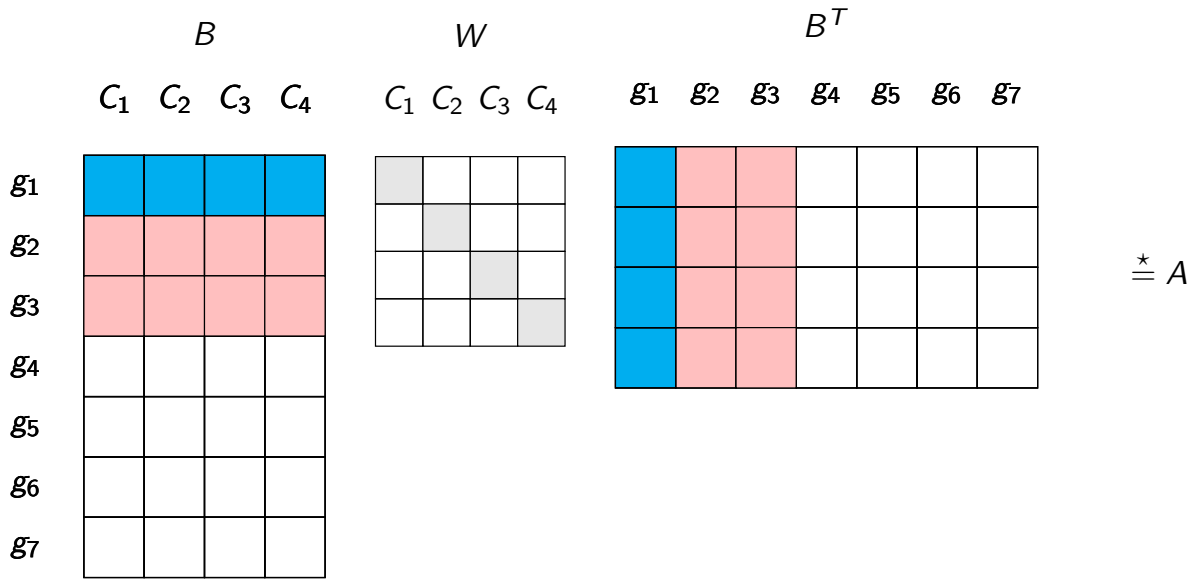
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Algorithm Overview



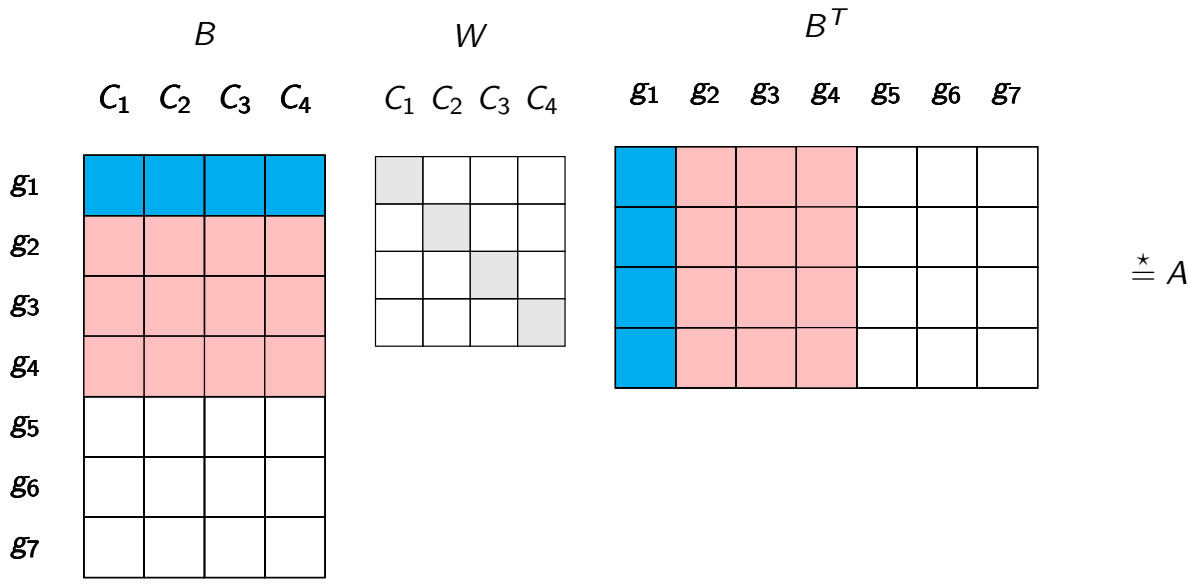
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Algorithm Overview



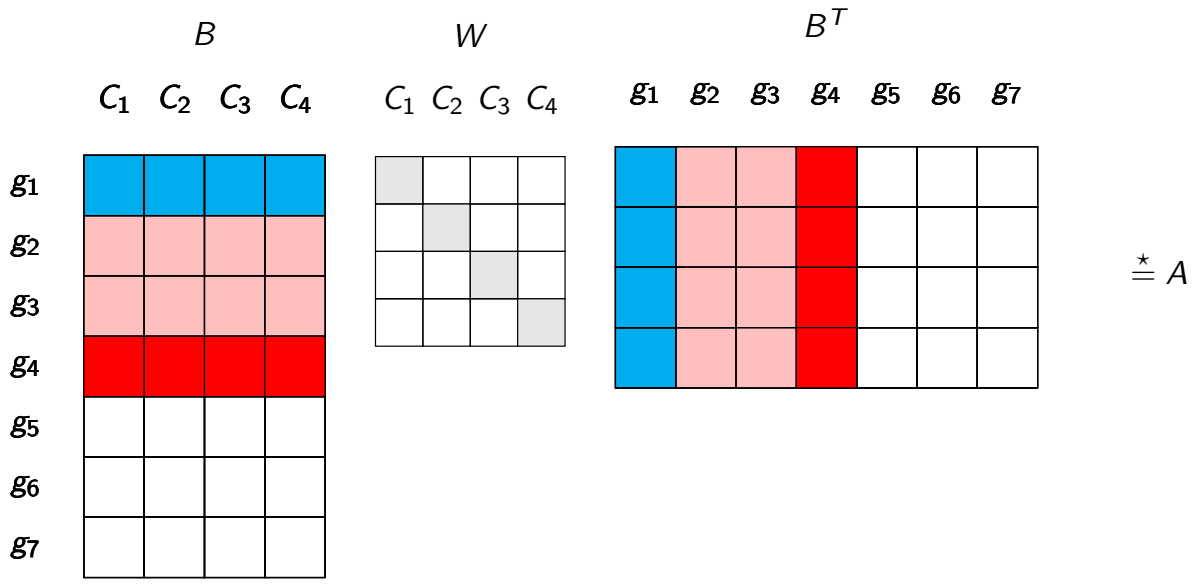
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Algorithm Overview



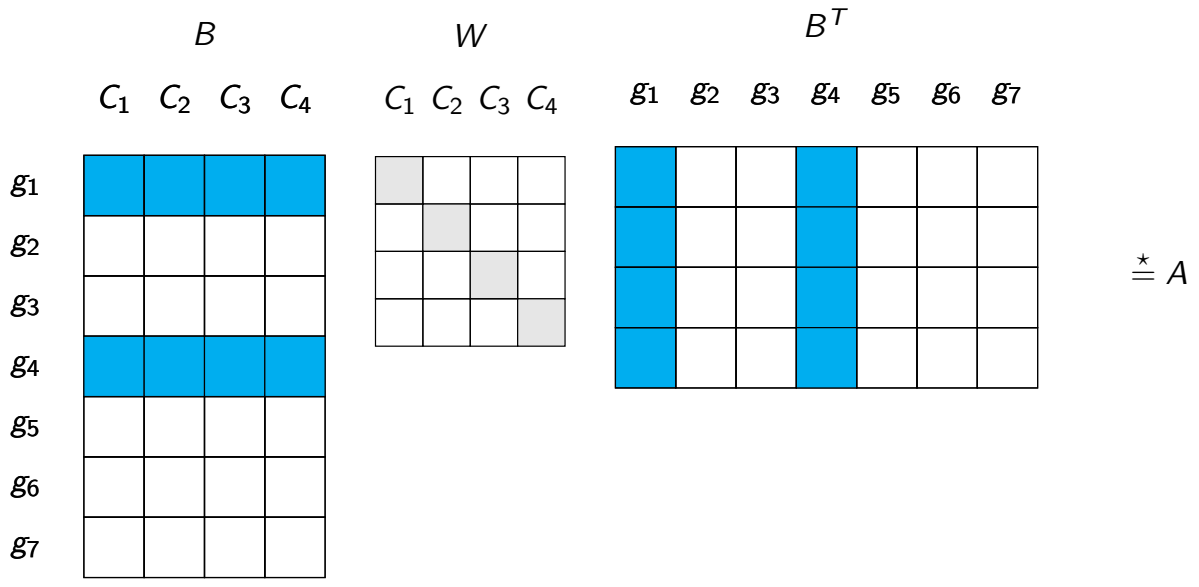
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Algorithm Overview



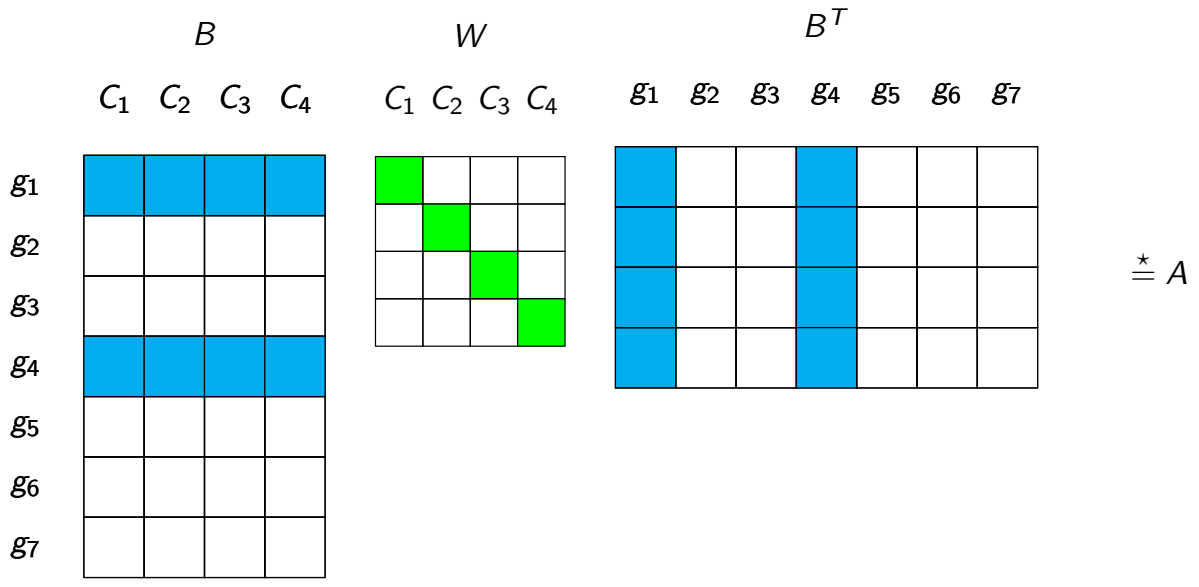
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Algorithm Overview



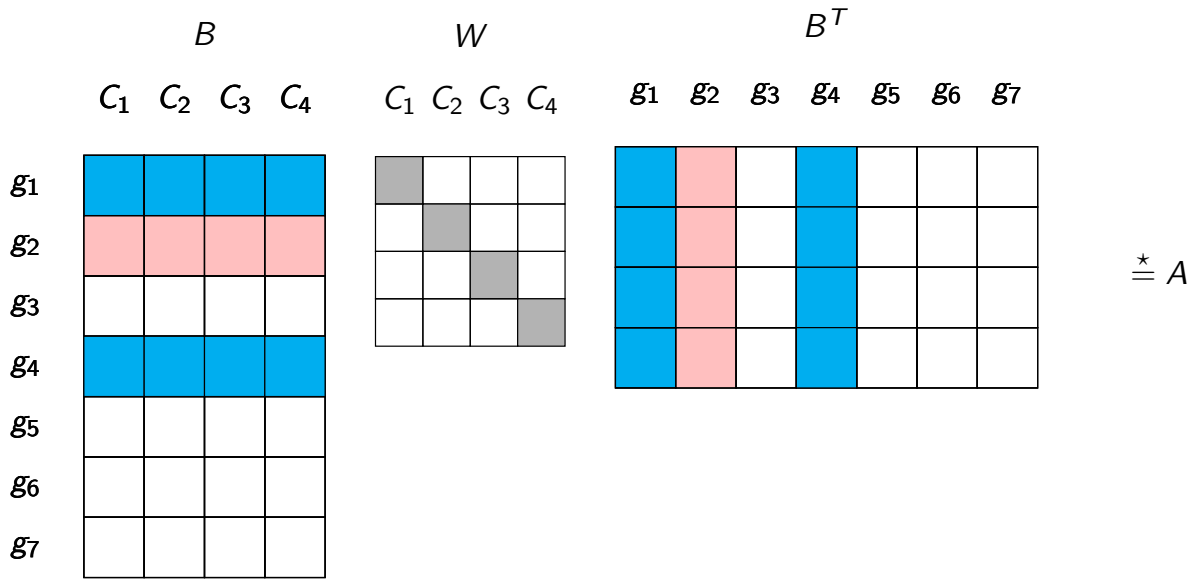
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Algorithm Overview



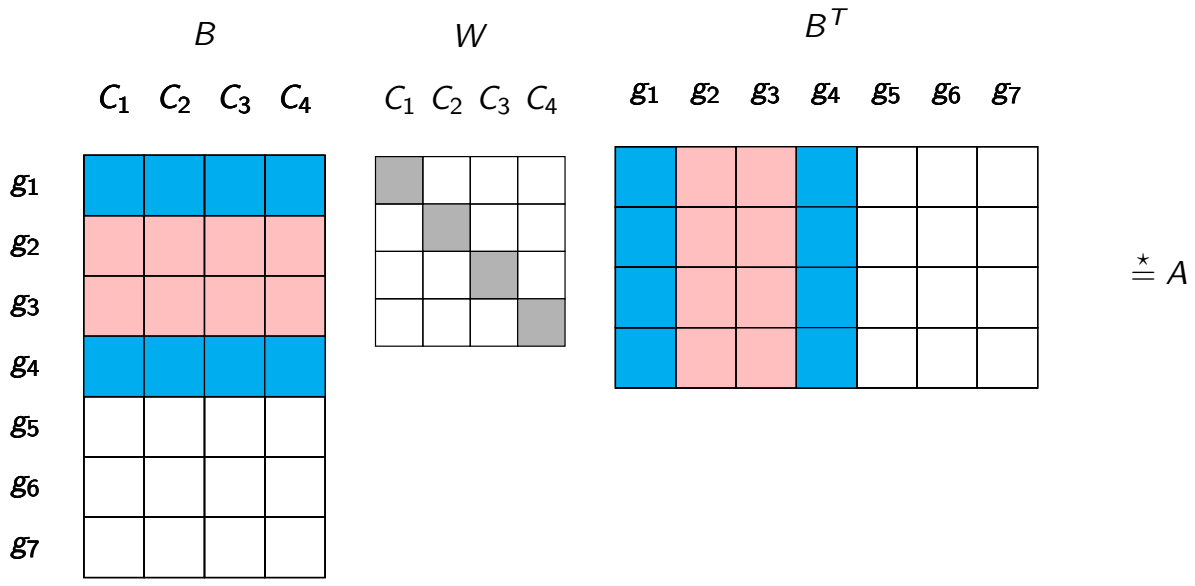
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Algorithm Overview



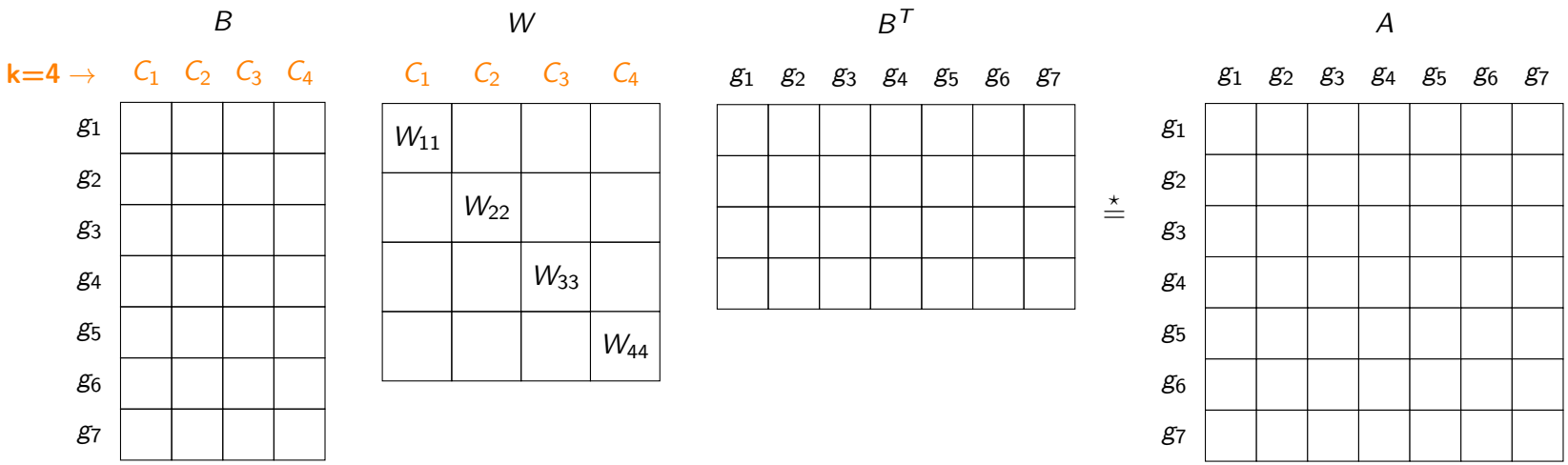
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Algorithm Overview



1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Linear Programming Weight Recovery



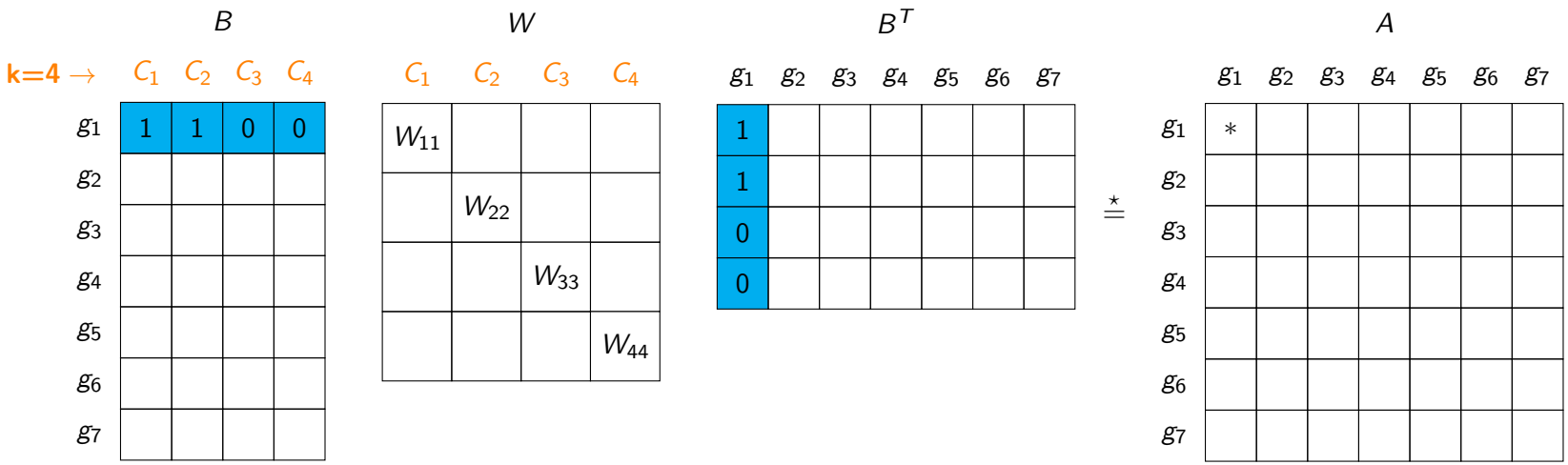
Linear Program

Find feasible $W_{11}, \ldots W_{kk}$ s.t.

- $W_{11} \geq 0, \ldots W_{kk} \geq 0$

Linear Programming Weight Recovery

1. guess basis row



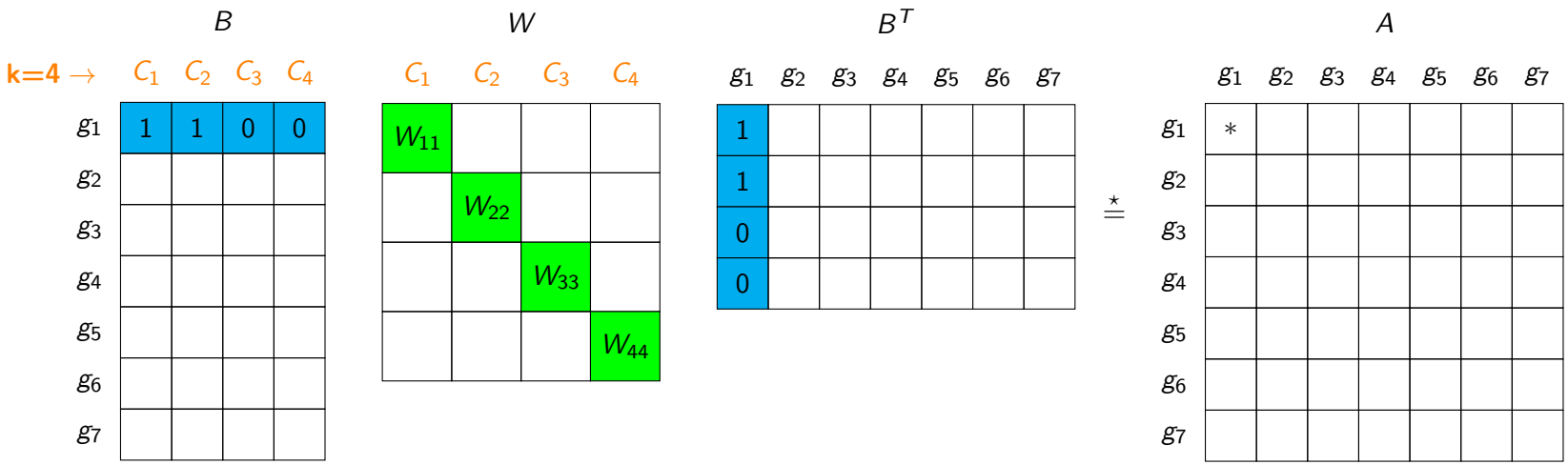
Linear Program

Find feasible $W_{11}, \ldots W_{kk}$ s.t.

- $W_{11} \geq 0, \ldots W_{kk} \geq 0$

Linear Programming Weight Recovery

2. solve for clique weights



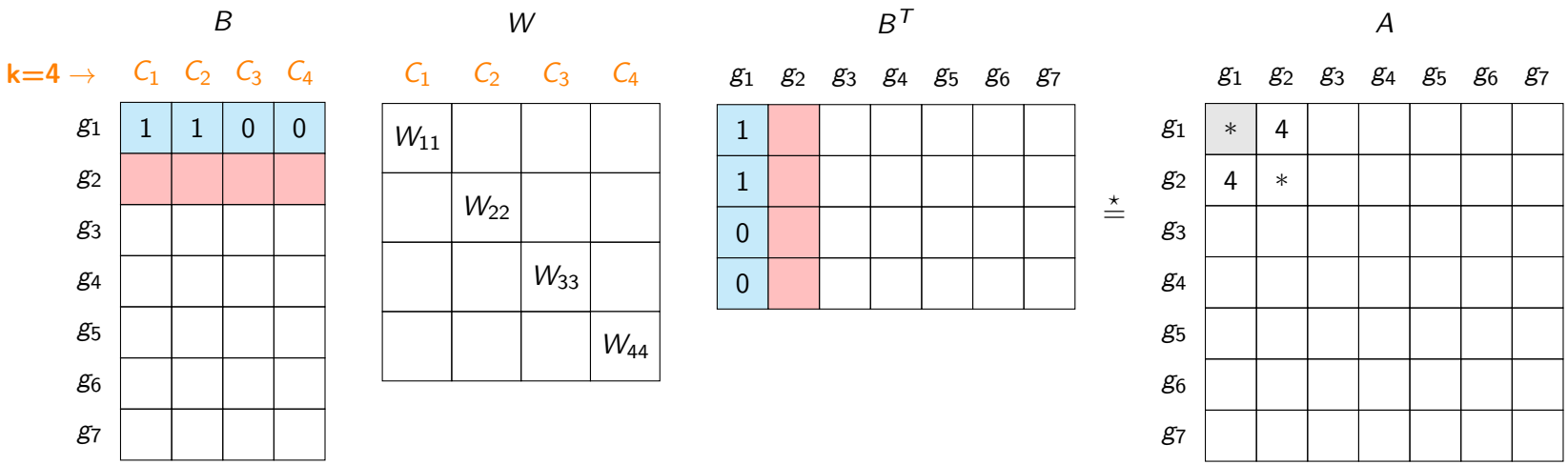
Linear Program

Find feasible $W_{11}, \ldots W_{kk}$ s.t.

- $W_{11} \geq 0, \ \ldots W_{kk} \geq 0$
- $\widetilde{B}_i^T W \widetilde{B}_i = A_{ii}$ if $A_{ii} \neq *$

Linear Programming Weight Recovery

3. iteratively fill in non-basis rows



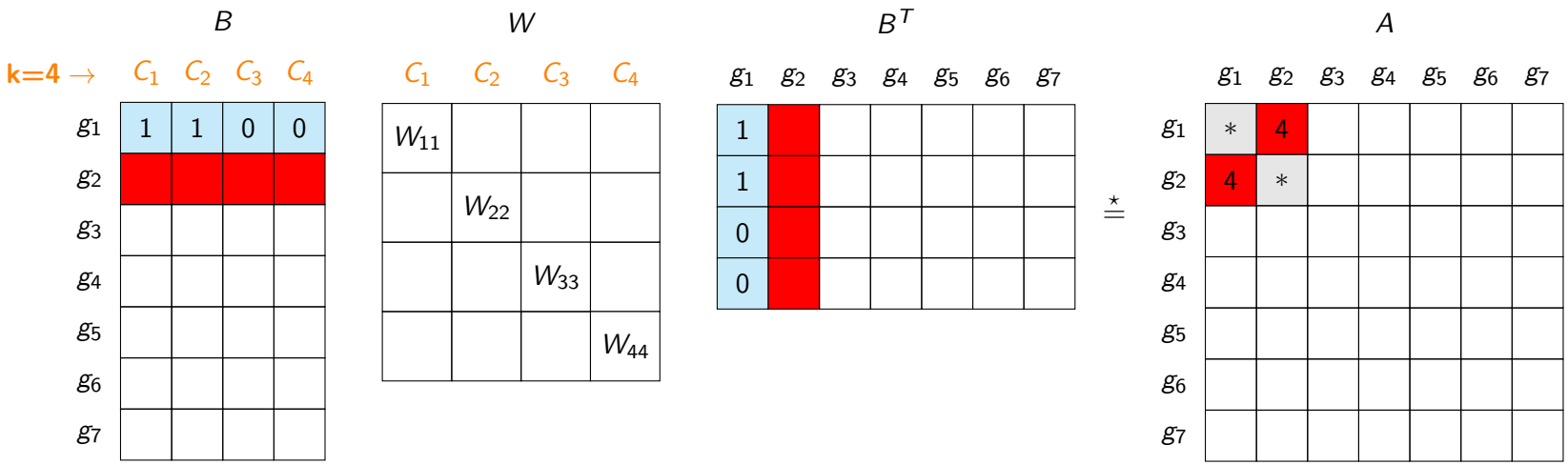
Linear Program

Find feasible $W_{11}, \ldots W_{kk}$ s.t.

- $W_{11} \geq 0, \ \ldots W_{kk} \geq 0$
- $\widetilde{B}_i^T W \widetilde{B}_i = A_{ii}$ if $A_{ii} \neq *$

Linear Programming Weight Recovery

incompatible non-basis row



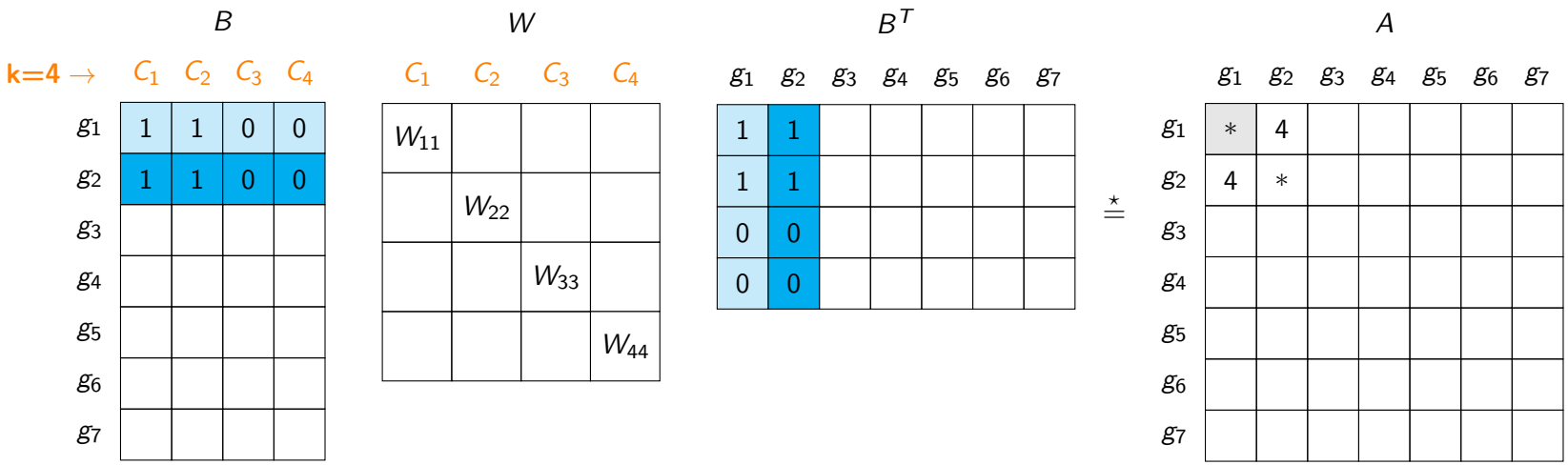
Linear Program

Find feasible $W_{11}, \ldots W_{kk}$ s.t.

- $W_{11} \geq 0, \ \ldots W_{kk} \geq 0$
- $\widetilde{B}_i^T W \widetilde{B}_i = A_{ii}$ if $A_{ii} \neq *$

Linear Programming Weight Recovery

1. guess basis row



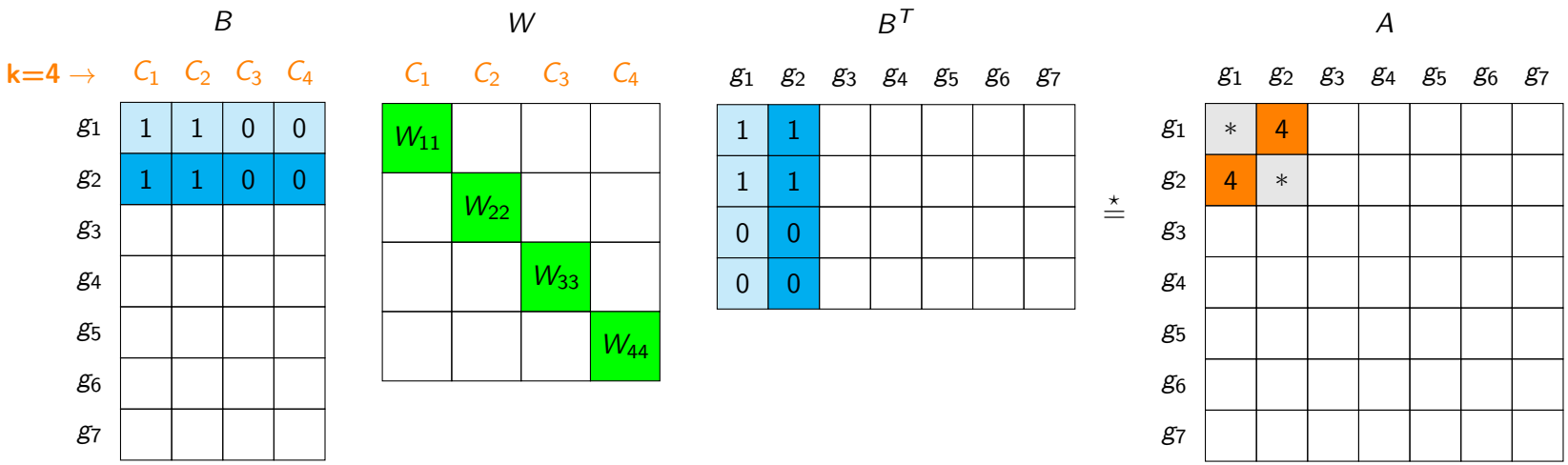
Linear Program

Find feasible $W_{11}, \ldots W_{kk}$ s.t.

- $W_{11} \geq 0, \ \ldots W_{kk} \geq 0$
- $\widetilde{B}_i^T W \widetilde{B}_i = A_{ii}$ if $A_{ii} \neq *$

Linear Programming Weight Recovery

2. solve for clique weights



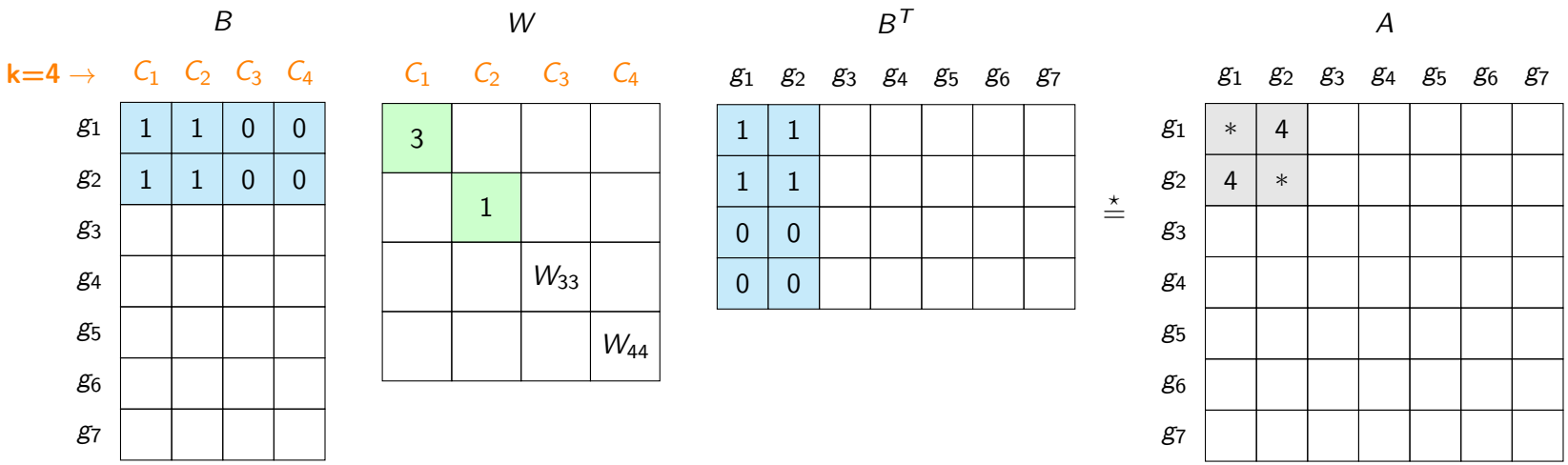
Linear Program

Find feasible $W_{11}, \ldots W_{kk}$ s.t.

- $W_{11} \geq 0, \ldots W_{kk} \geq 0$
- $\widetilde{B}_i^T W \widetilde{B}_i = A_{ii}$ if $A_{ii} \neq *$
- $\widetilde{B}_i^T W \widetilde{B}_j = A_{ij}$ for each distinct $\widetilde{B}_i$, $\widetilde{B}_j$

Linear Programming Weight Recovery

2. solve for clique weights



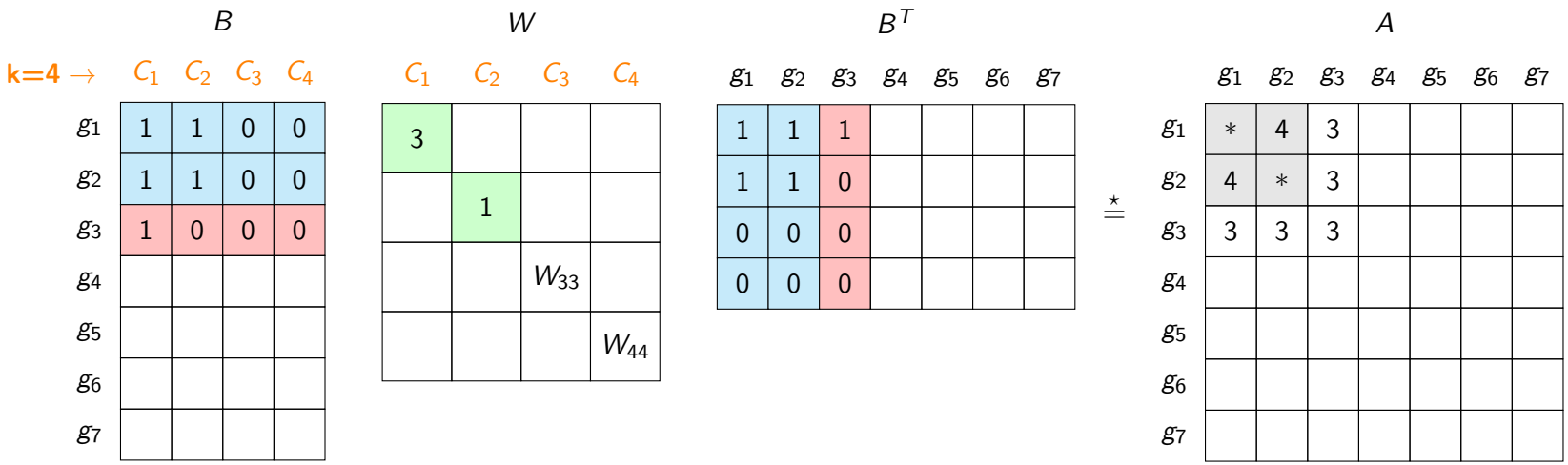
Linear Program

Find feasible $W_{11}, \ldots W_{kk}$ s.t.

- $W_{11} \geq 0, \ \ldots W_{kk} \geq 0$
- $\widetilde{B}_i^T W \widetilde{B}_i = A_{ii}$ if $A_{ii} \neq *$
- $\widetilde{B}_i^T W \widetilde{B}_j = A_{ij}$ for each distinct $\widetilde{B}_i$, $\widetilde{B}_j$

Linear Programming Weight Recovery

3. iteratively fill in non-basis rows



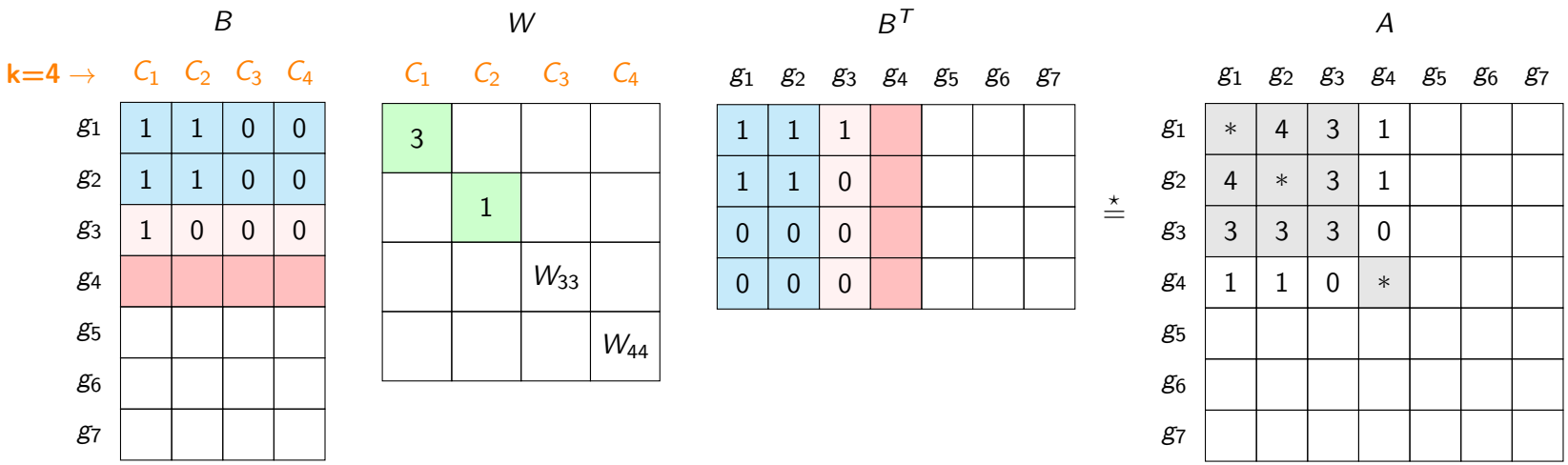
Linear Program

Find feasible $W_{11}, \ldots W_{kk}$ s.t.

- $W_{11} \geq 0, \ldots W_{kk} \geq 0$
- $\widetilde{B}_i^T W \widetilde{B}_i = A_{ii}$ if $A_{ii} \neq *$
- $\widetilde{B}_i^T W \widetilde{B}_j = A_{ij}$ for each distinct $\widetilde{B}_i$, $\widetilde{B}_j$

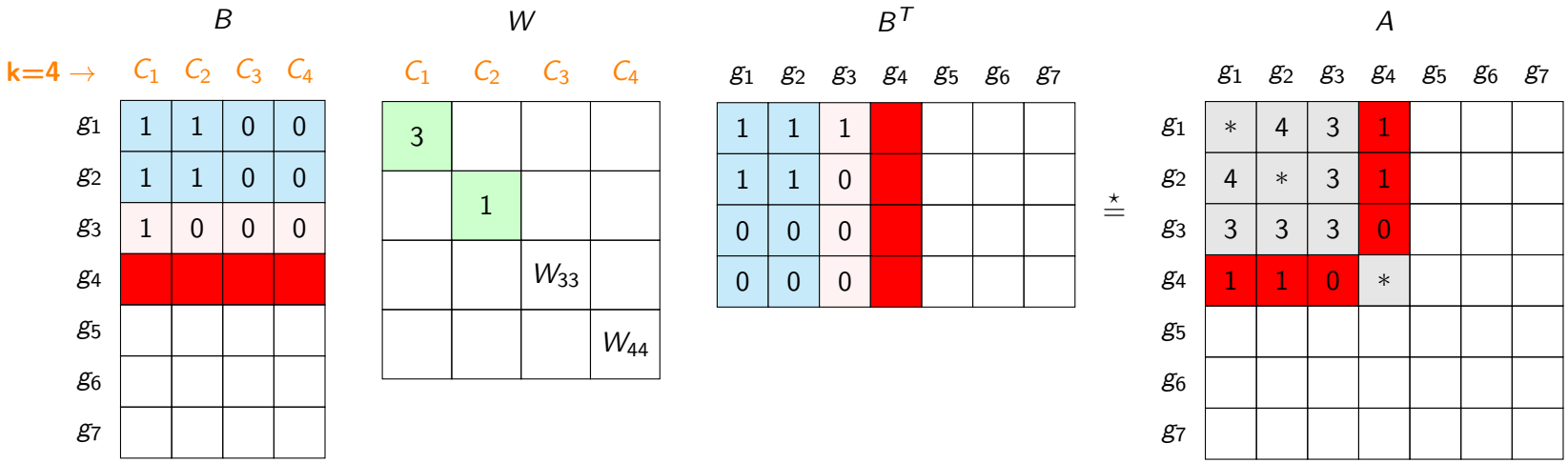
Linear Programming Weight Recovery

3. iteratively fill in non-basis rows



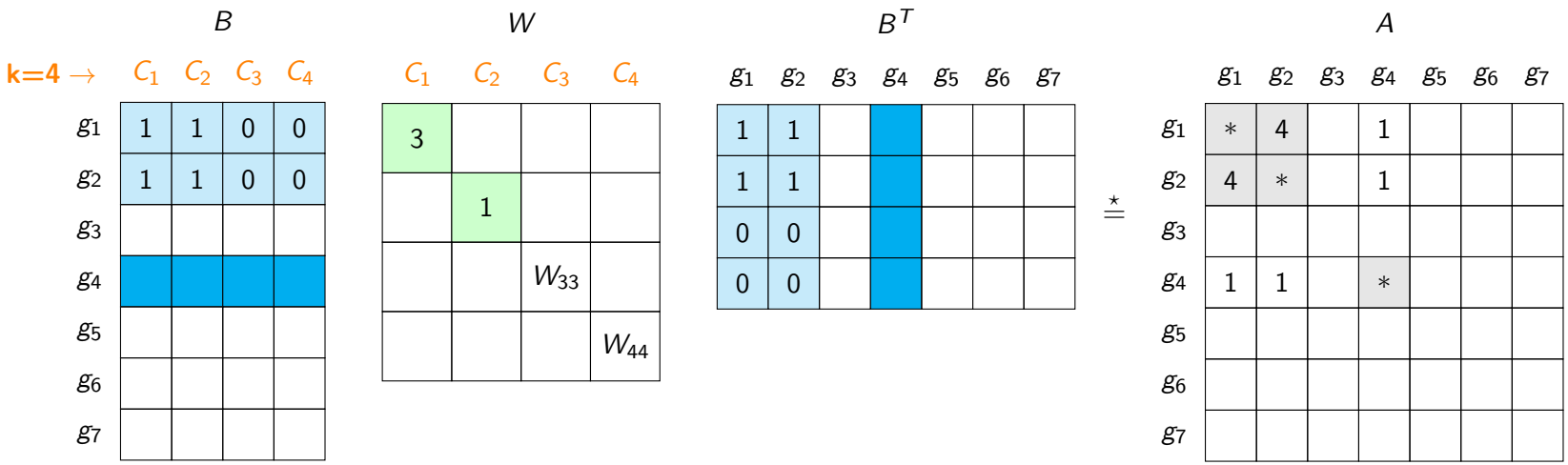
Linear Programming Weight Recovery

incompatible non-basis row



Linear Programming Weight Recovery

1. guess basis row

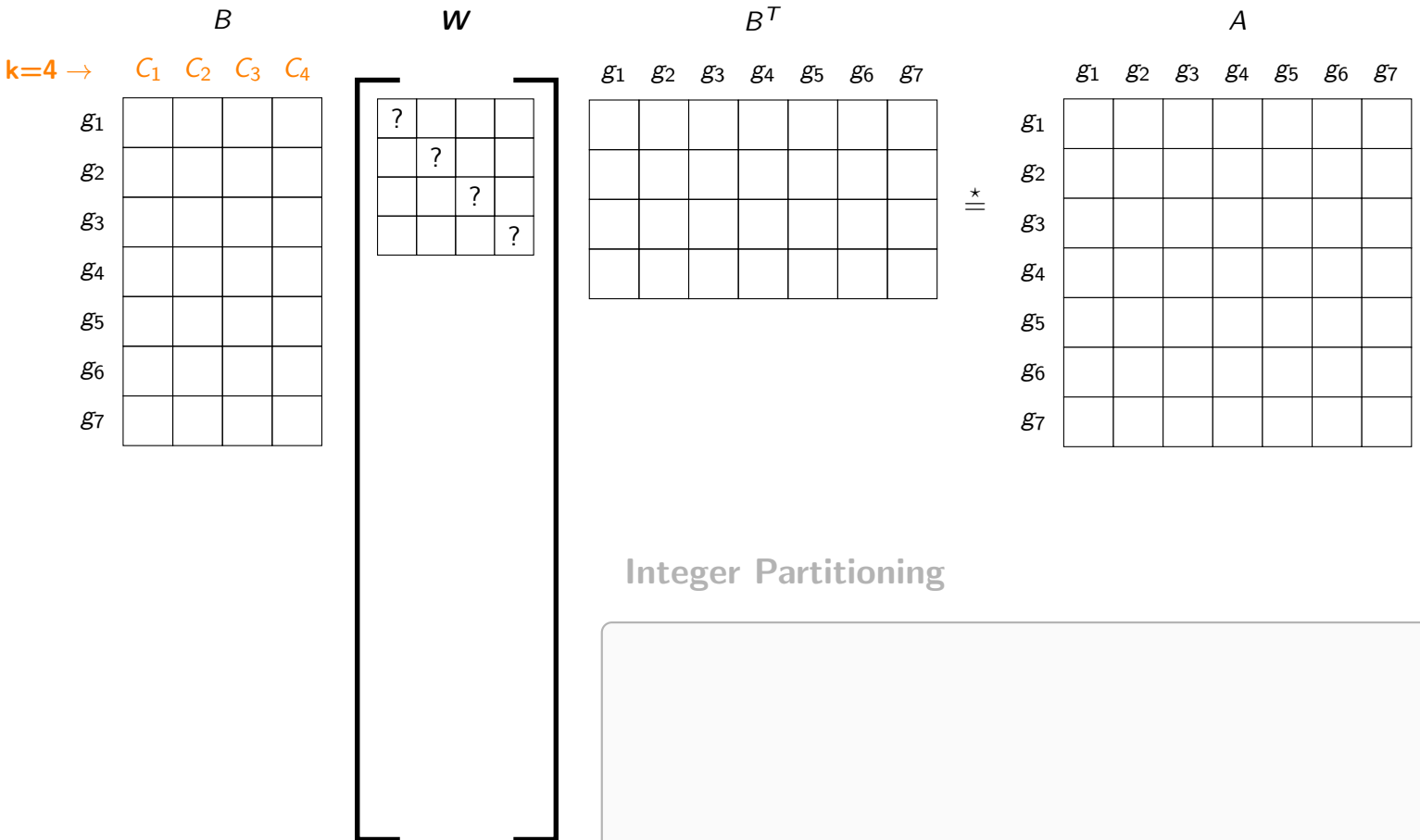


Linear Program

Find feasible $W_{11}, \ldots W_{kk}$ s.t.

- $W_{11} \geq 0, \ldots W_{kk} \geq 0$
- $\widetilde{B}_i^T W \widetilde{B}_i = A_{ii}$ if $A_{ii} \neq *$
- $\widetilde{B}_i^T W \widetilde{B}_j = A_{ij}$ for each distinct $\widetilde{B}_i$, $\widetilde{B}_j$

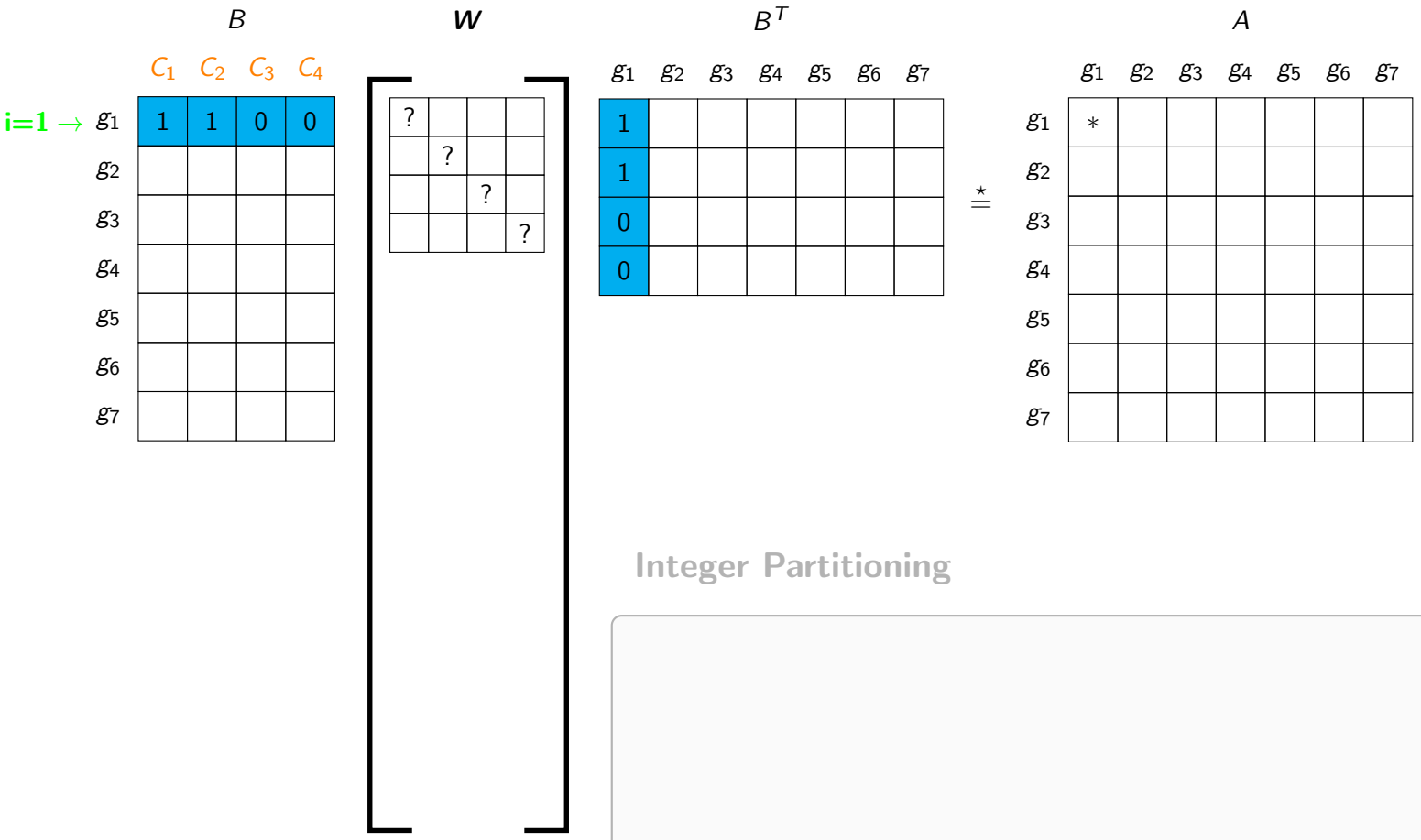
Integer Partitioning Weight Recovery



Integer Partitioning

Integer Partitioning Weight Recovery

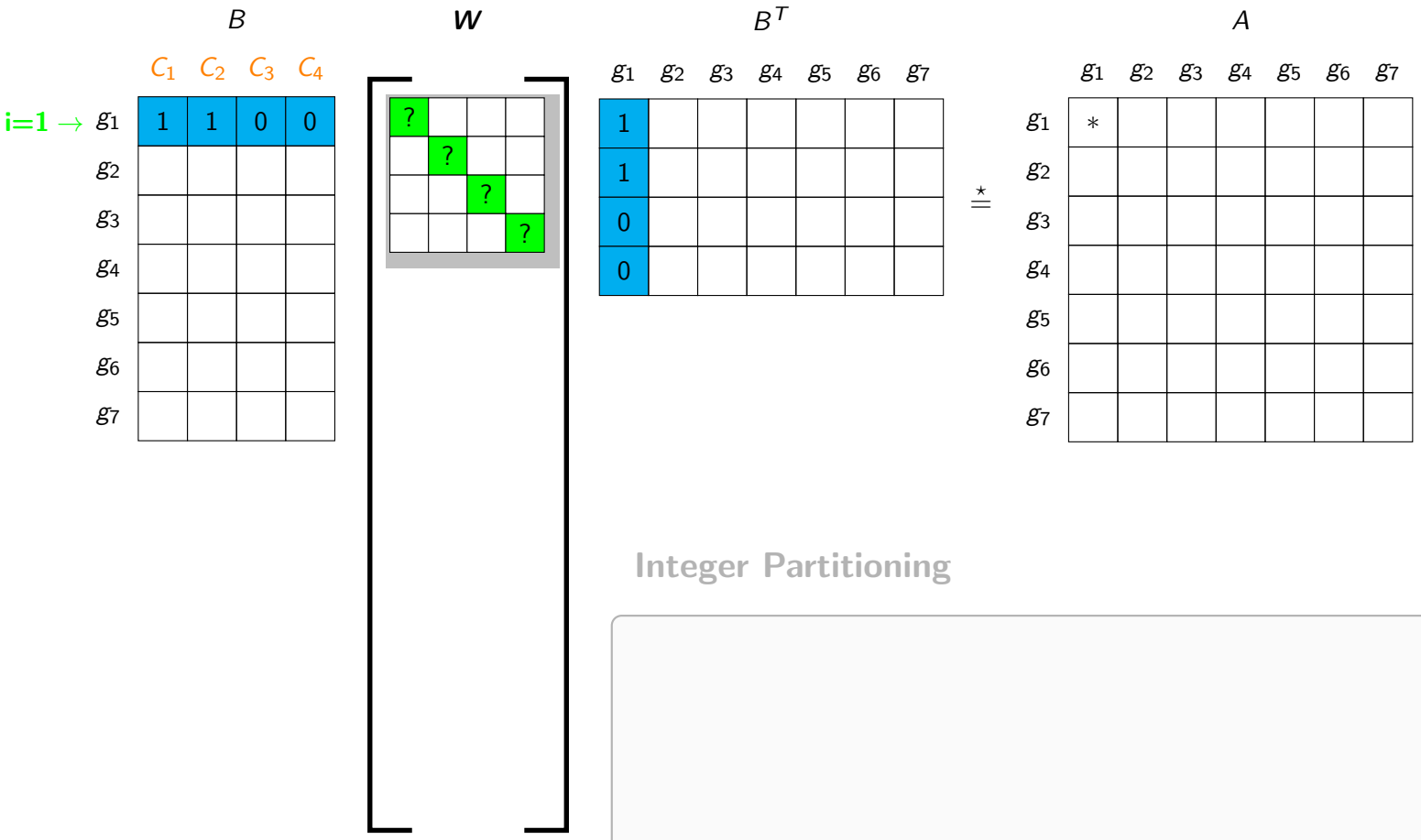
1. guess basis row



Integer Partitioning

Integer Partitioning Weight Recovery

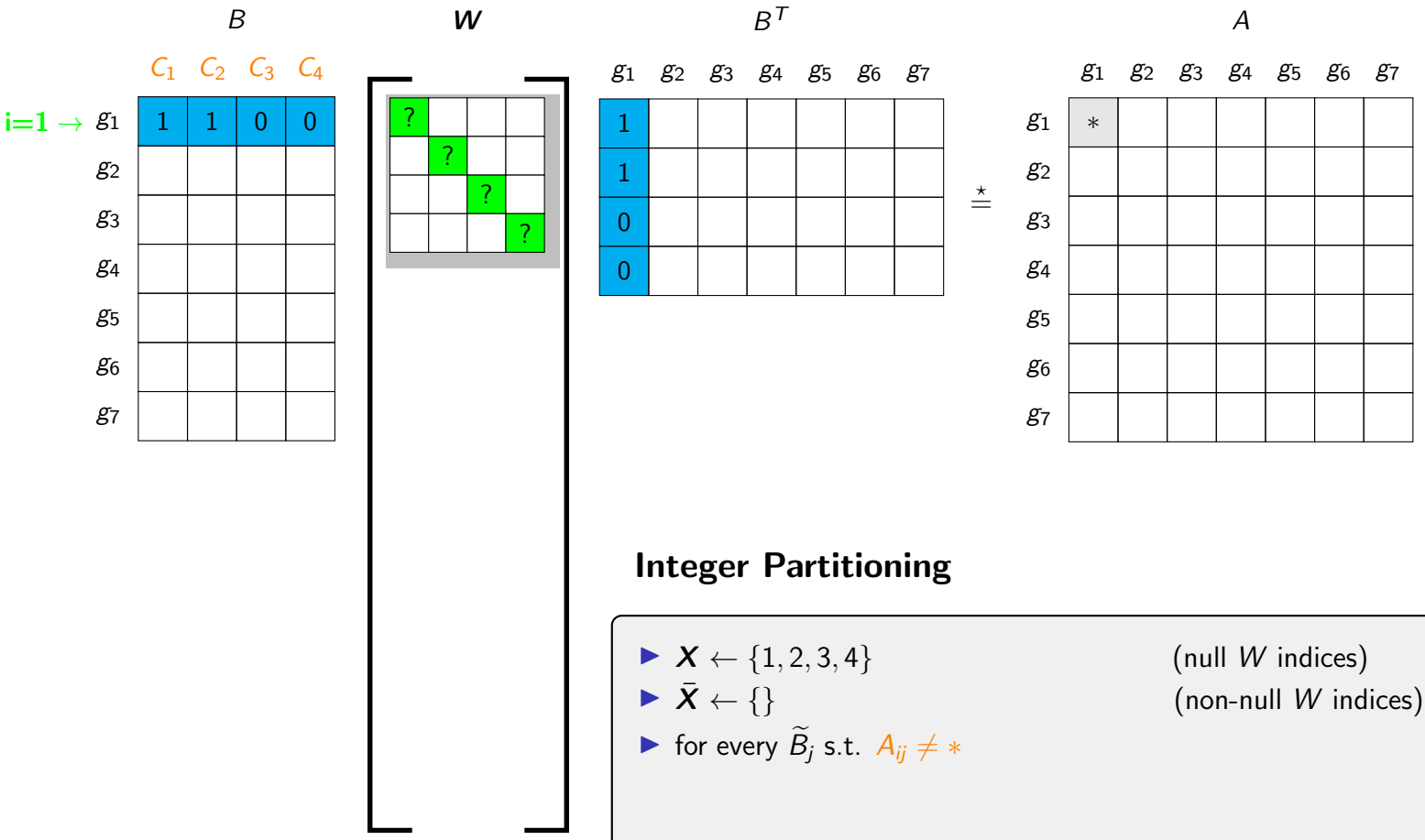
2. solve for clique weights



Integer Partitioning

Integer Partitioning Weight Recovery

2. solve for clique weights

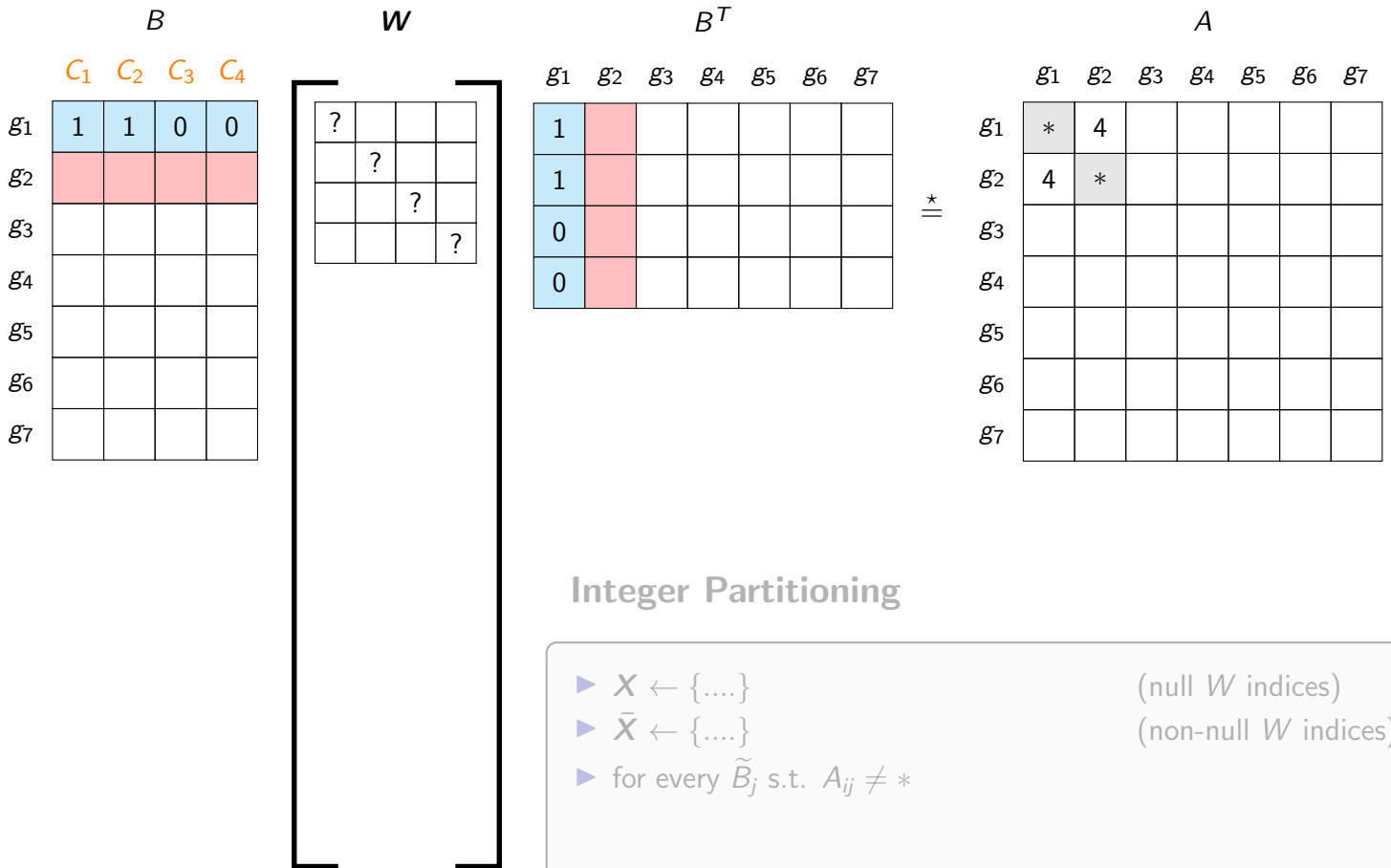


Integer Partitioning

- $\boldsymbol{X} \leftarrow \{1, 2, 3, 4\}$ (null W indices)
- $\bar{\boldsymbol{X}} \leftarrow \{\}$ (non-null W indices)
- for every $\widetilde{B}_j$ s.t. $A_{ij} \neq *$

Integer Partitioning Weight Recovery

3. iteratively fill in non-basis rows

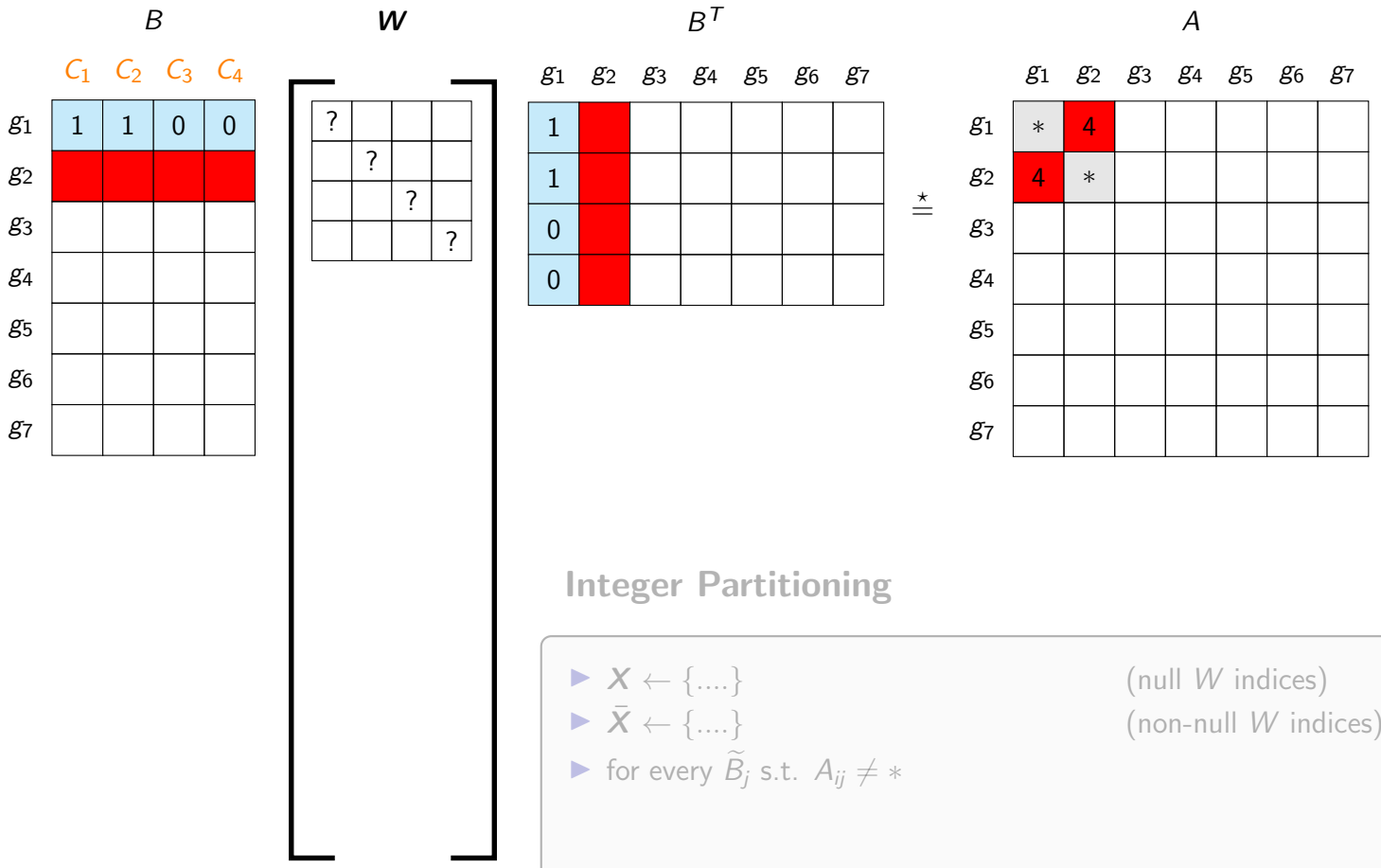


Integer Partitioning

- $X \leftarrow \{....\}$ (null W indices)
- $\bar{X} \leftarrow \{....\}$ (non-null W indices)
- for every $\widetilde{B}_j$ s.t. $A_{ij} \neq *$

Integer Partitioning Weight Recovery

incompatible non-basis row

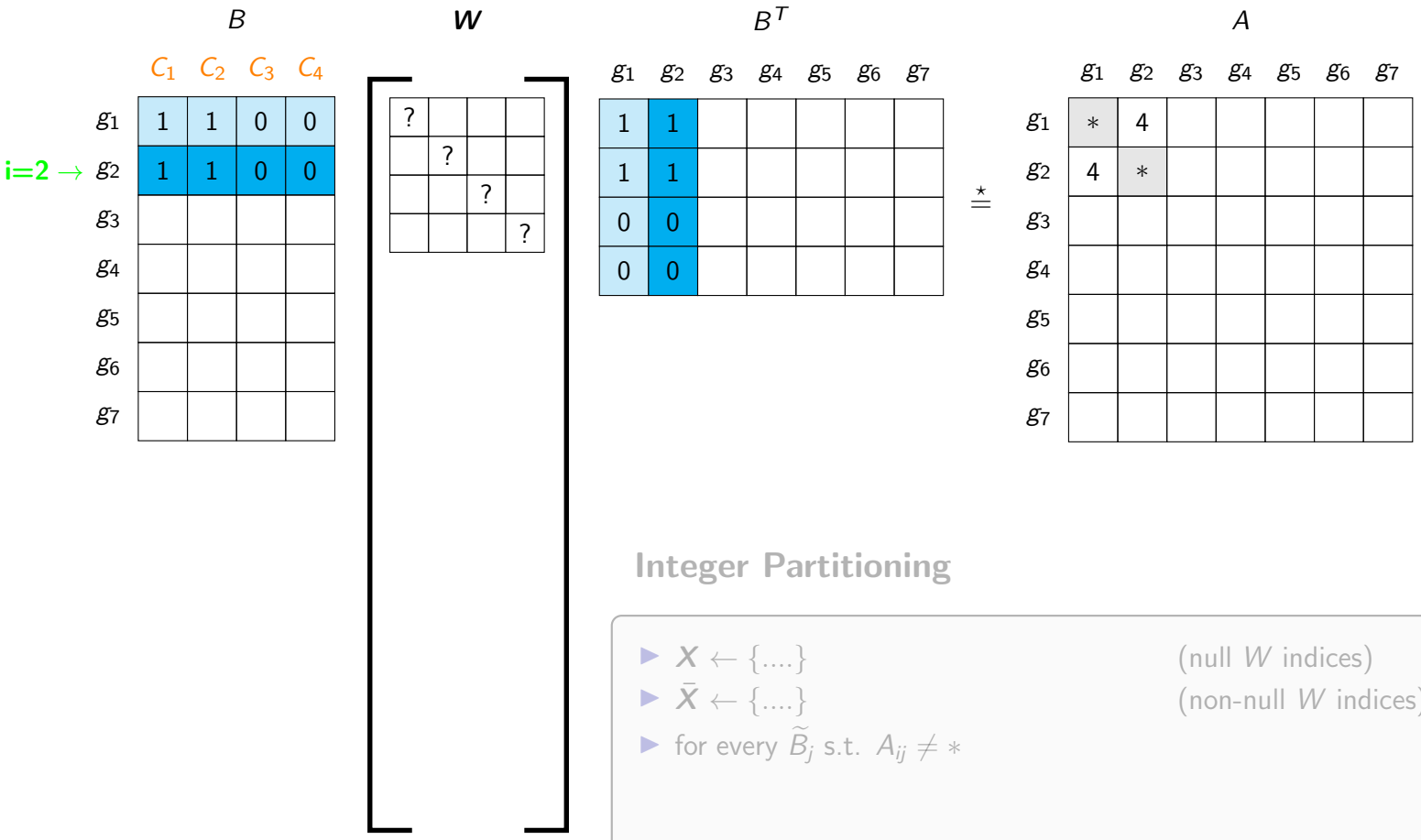


Integer Partitioning

- $X \leftarrow \{....\}$ (null W indices)
- $\bar{X} \leftarrow \{....\}$ (non-null W indices)
- for every $\widetilde{B}_j$ s.t. $A_{ij} \neq *$

Integer Partitioning Weight Recovery

1. guess basis row

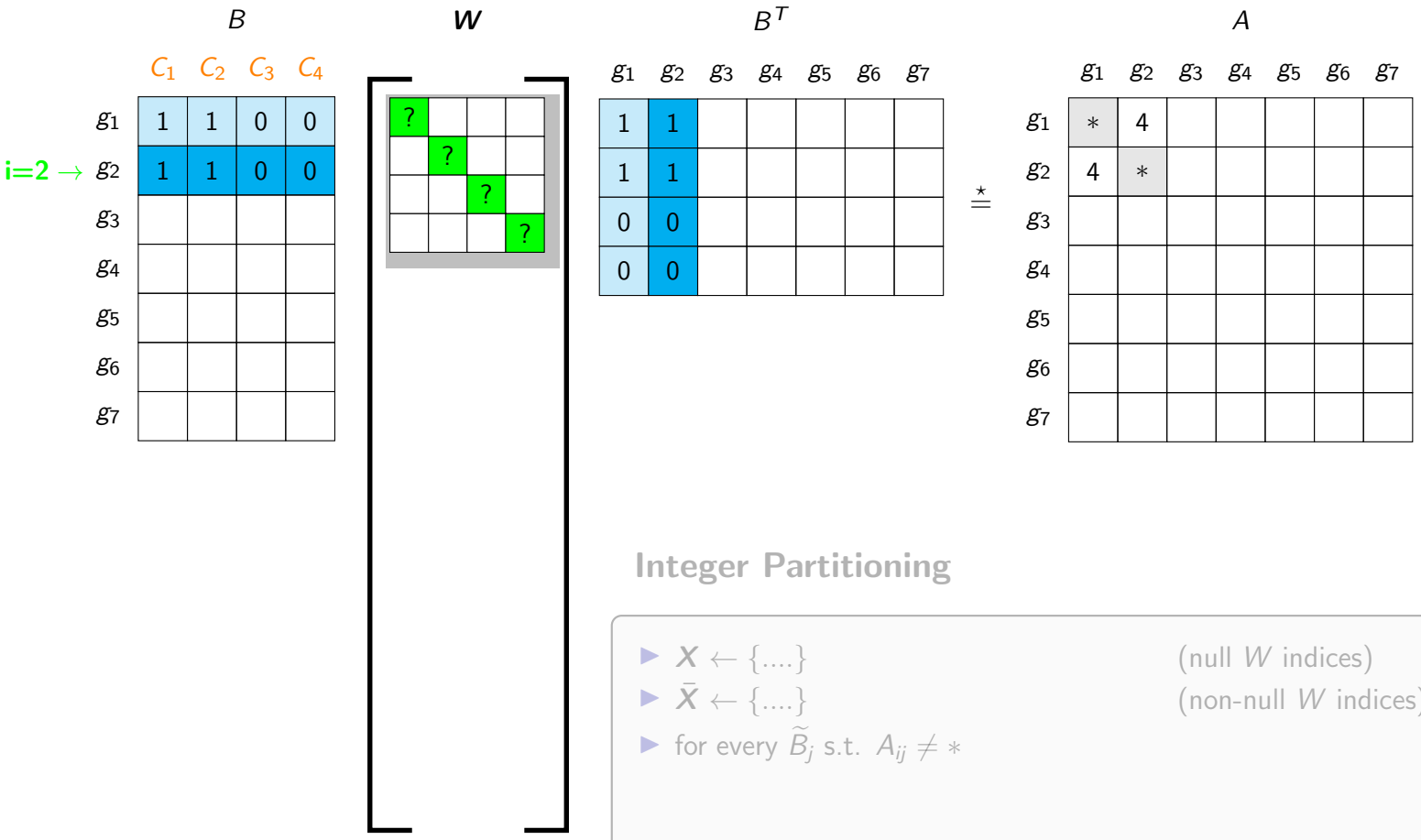


Integer Partitioning

- $X \leftarrow \{....\}$ (null W indices)
- $\bar{X} \leftarrow \{....\}$ (non-null W indices)
- for every $\widetilde{B}_j$ s.t. $A_{ij} \neq *$

Integer Partitioning Weight Recovery

2. solve for clique weights

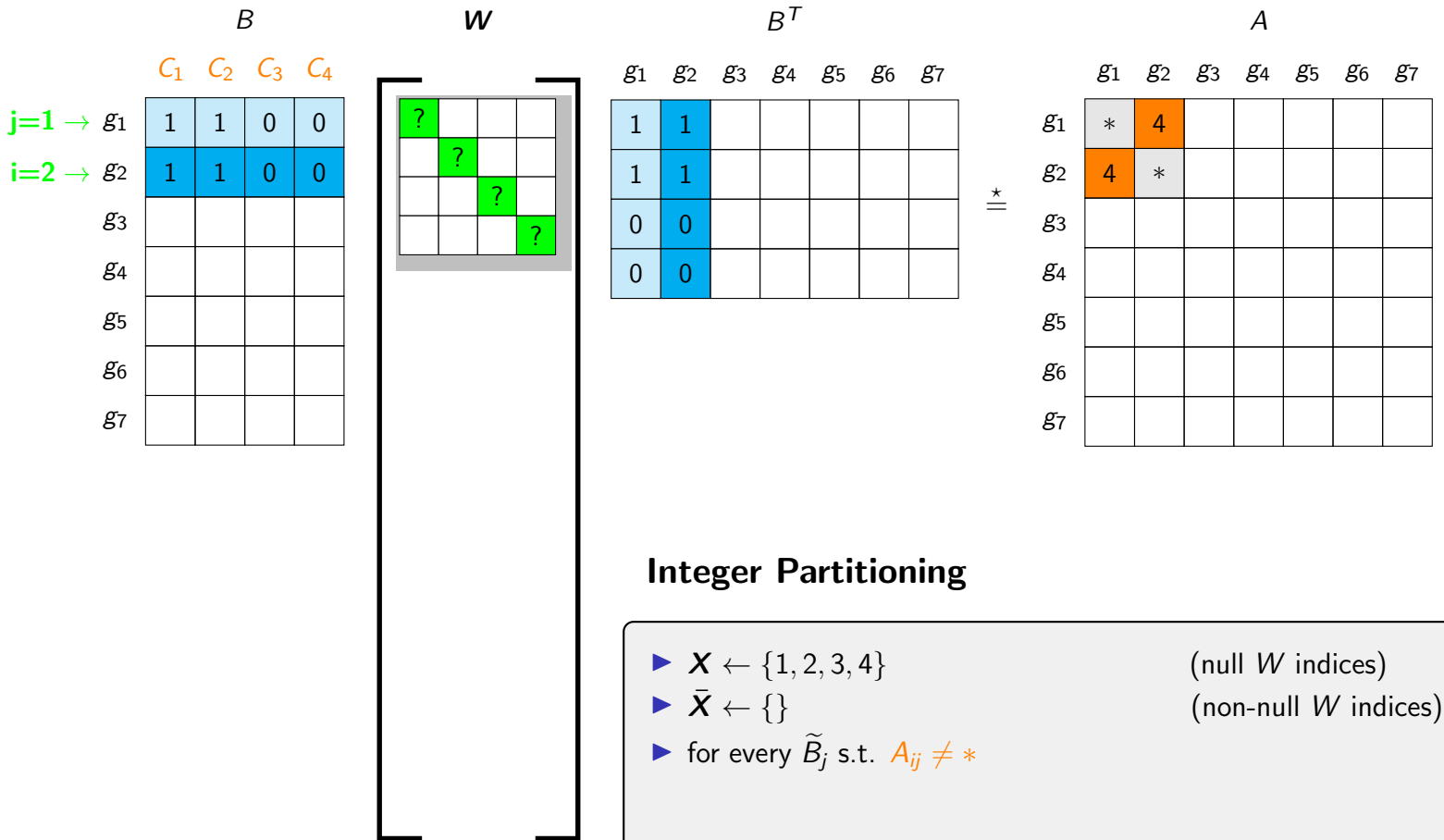


Integer Partitioning

- $X \leftarrow \{....\}$ (null W indices)
- $\bar{X} \leftarrow \{....\}$ (non-null W indices)
- for every $\widetilde{B}_j$ s.t. $A_{ij} \neq *$

Integer Partitioning Weight Recovery

2. solve for clique weights

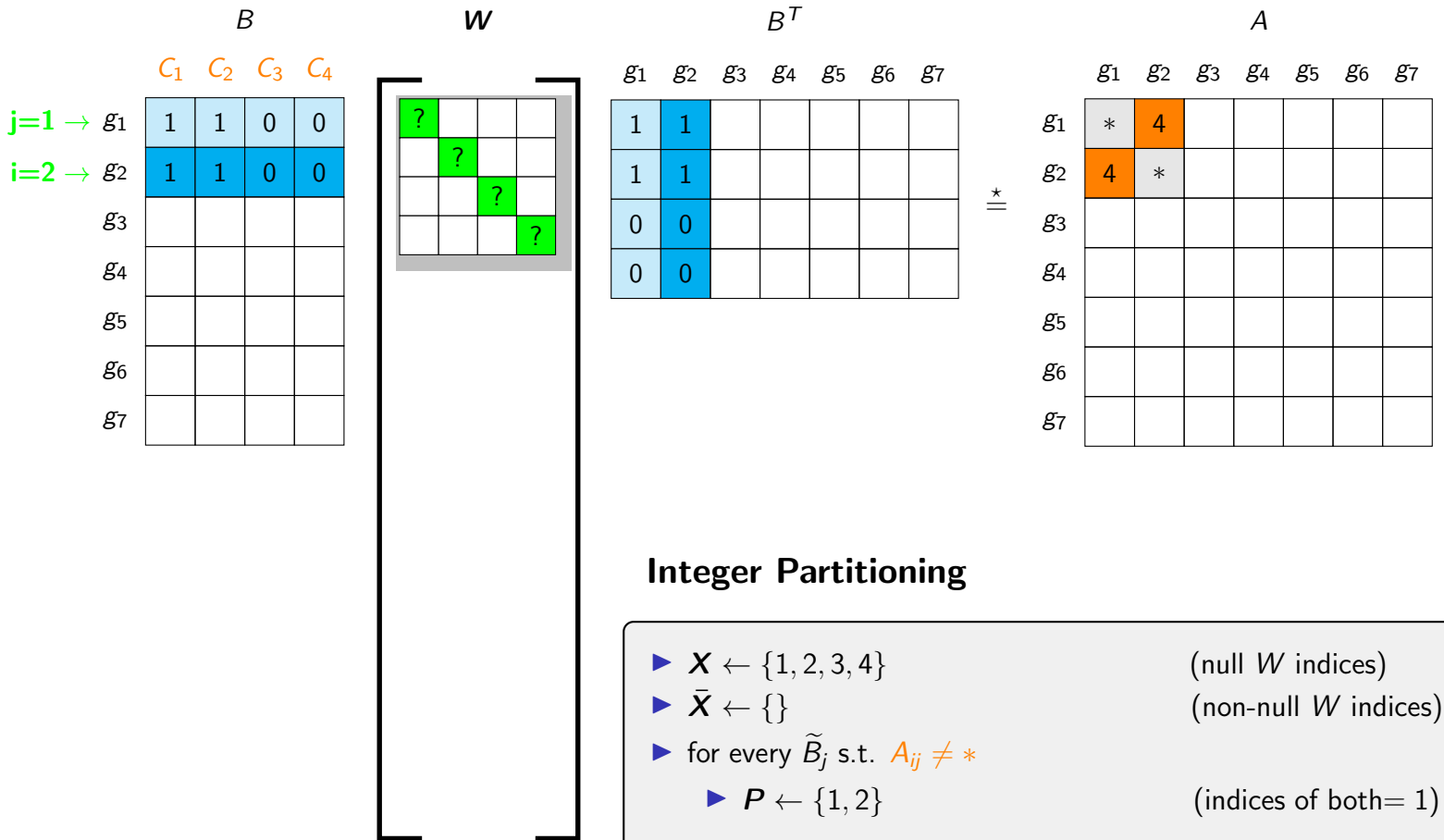


Integer Partitioning

- $\boldsymbol{X} \leftarrow \{1, 2, 3, 4\}$ (null W indices)
- $\bar{\boldsymbol{X}} \leftarrow \{\}$ (non-null W indices)
- for every $\widetilde{B}_j$ s.t. $A_{ij} \neq *$

Integer Partitioning Weight Recovery

2. solve for clique weights

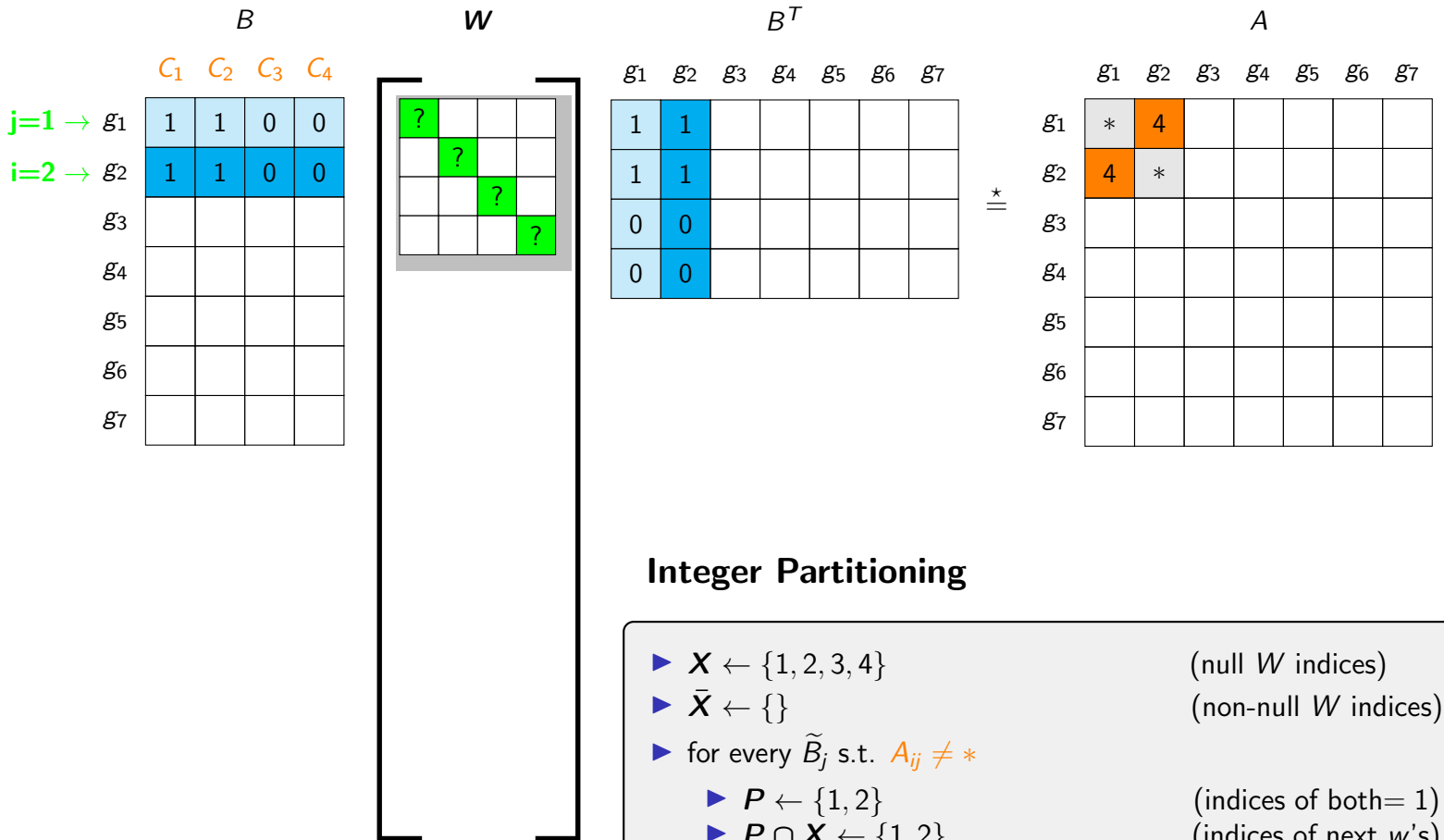


Integer Partitioning

- $\boldsymbol{X} \leftarrow \{1, 2, 3, 4\}$ (null W indices)
- $\bar{\boldsymbol{X}} \leftarrow \{\}$ (non-null W indices)
- for every $\widetilde{B}_j$ s.t. $A_{ij} \neq *$
  - $\boldsymbol{P} \leftarrow \{1, 2\}$ (indices of both$=1$)

Integer Partitioning Weight Recovery

2. solve for clique weights

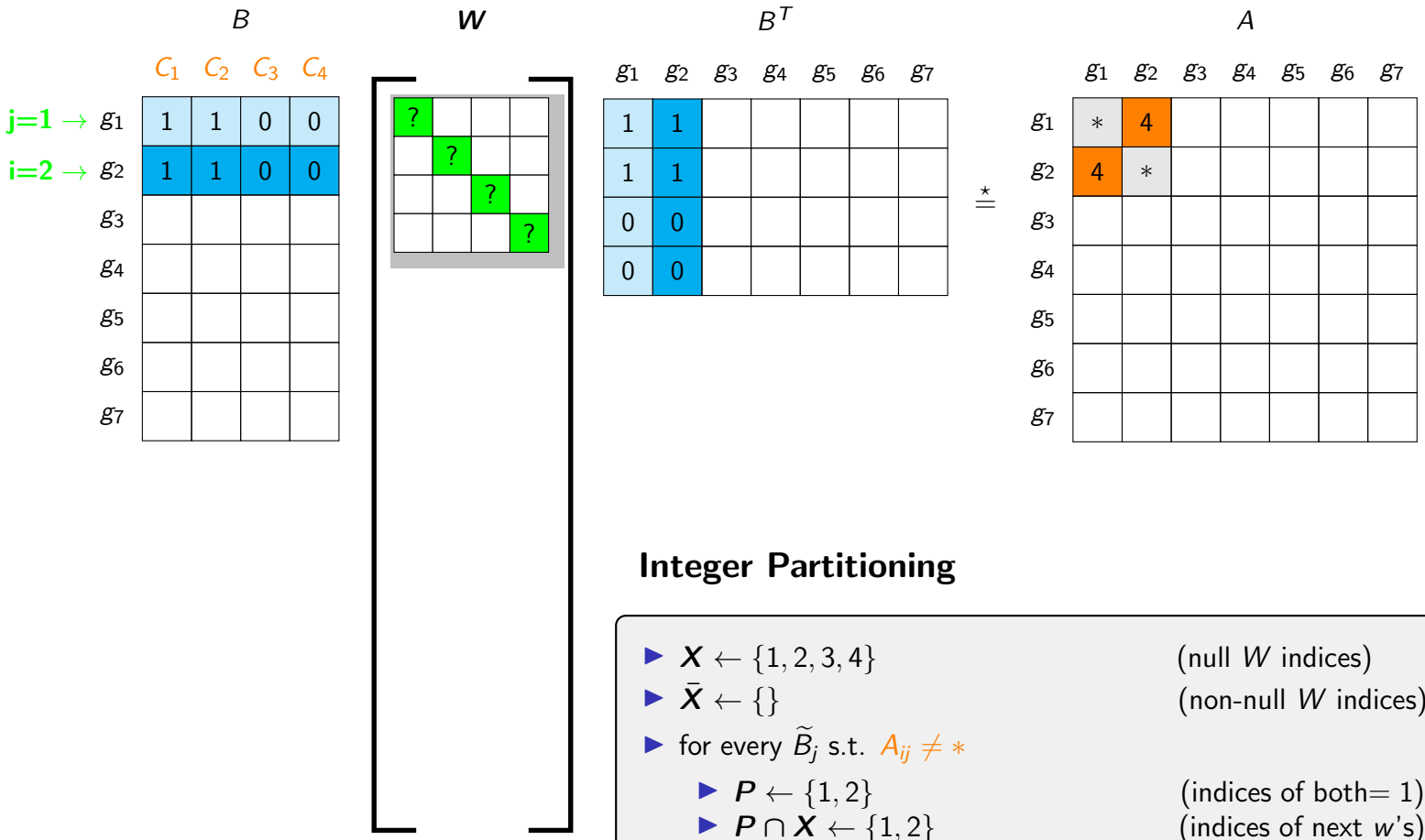


Integer Partitioning

- $\boldsymbol{X} \leftarrow \{1, 2, 3, 4\}$ (null W indices)
- $\bar{\boldsymbol{X}} \leftarrow \{\}$ (non-null W indices)
- for every $\widetilde{B}_j$ s.t. $A_{ij} \neq *$
  - $\boldsymbol{P} \leftarrow \{1, 2\}$ (indices of both$= 1$)
  - $\boldsymbol{P} \cap \boldsymbol{X} \leftarrow \{1, 2\}$ (indices of next w's)

Integer Partitioning Weight Recovery

2. solve for clique weights

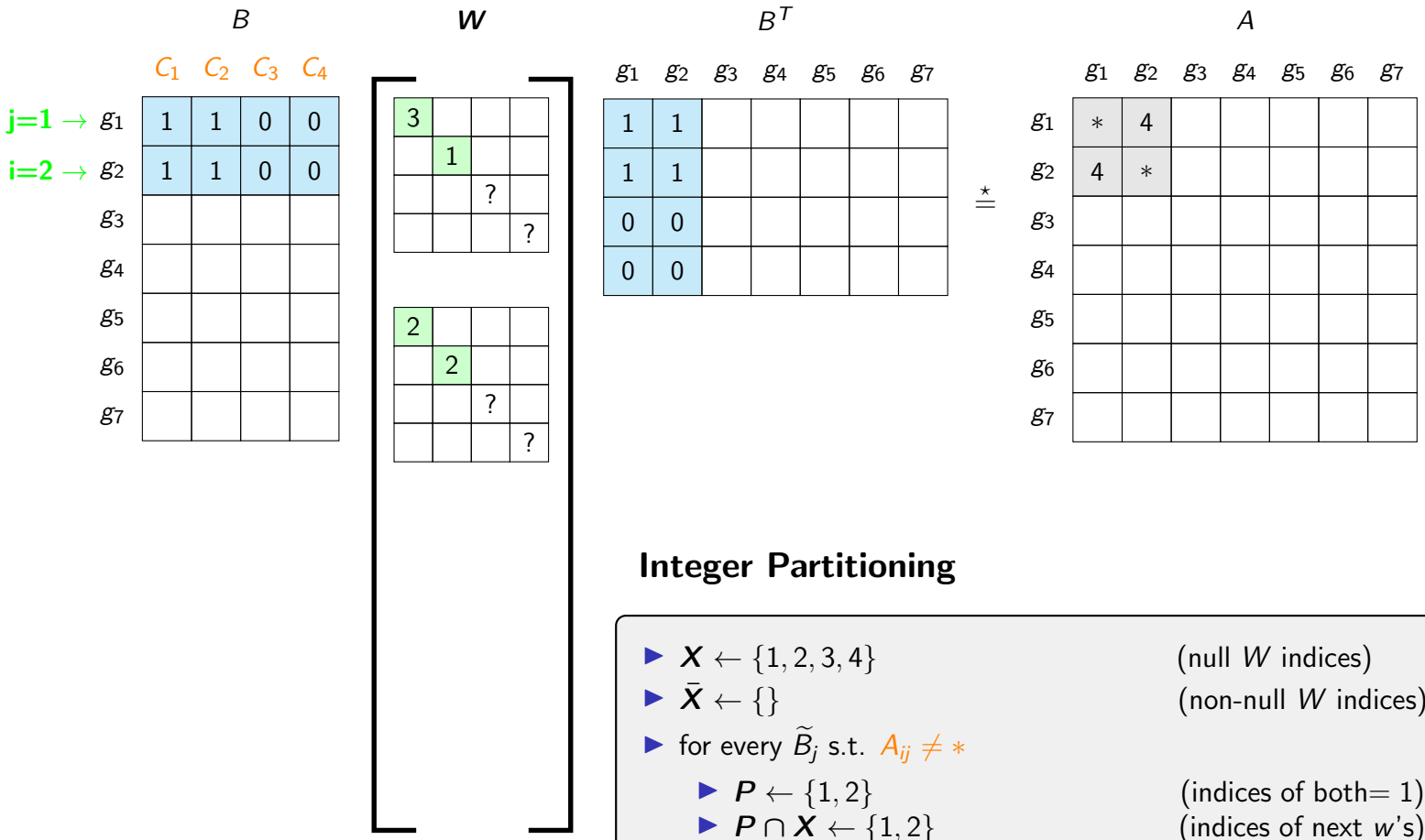


Integer Partitioning

- $\boldsymbol{X} \leftarrow \{1, 2, 3, 4\}$ (null W indices)
- $\bar{\boldsymbol{X}} \leftarrow \{\}$ (non-null W indices)
- for every $\widetilde{B}_j$ s.t. $A_{ij} \neq *$
  - $\boldsymbol{P} \leftarrow \{1, 2\}$ (indices of both$= 1$)
  - $\boldsymbol{P} \cap \boldsymbol{X} \leftarrow \{1, 2\}$ (indices of next w's)
  - $\{3, 1\}, \{2, 2\} \leftarrow$ partitions of $A_{ij} = 4$ with 2 integers

Integer Partitioning Weight Recovery

2. solve for clique weights

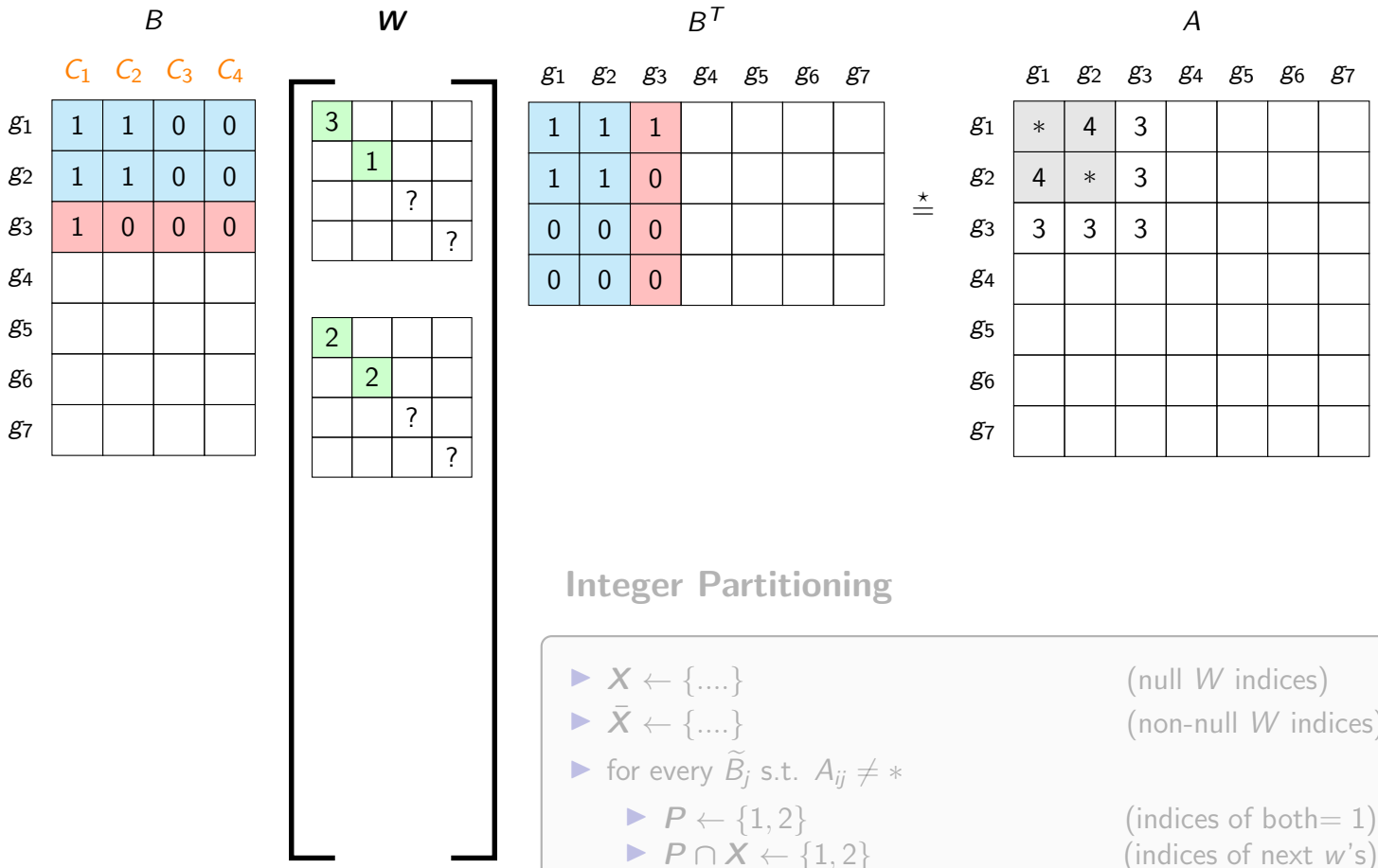


Integer Partitioning

- $\boldsymbol{X} \leftarrow \{1, 2, 3, 4\}$ (null W indices)
- $\bar{\boldsymbol{X}} \leftarrow \{\}$ (non-null W indices)
- for every $\widetilde{B}_j$ s.t. $A_{ij} \neq *$
  - $\boldsymbol{P} \leftarrow \{1, 2\}$ (indices of both$= 1$)
  - $\boldsymbol{P} \cap \boldsymbol{X} \leftarrow \{1, 2\}$ (indices of next w's)
  - $\{3, 1\}, \{2, 2\} \leftarrow$ partitions of $A_{ij} = 4$ with 2 integers

Integer Partitioning Weight Recovery

3. iteratively fill in non-basis rows

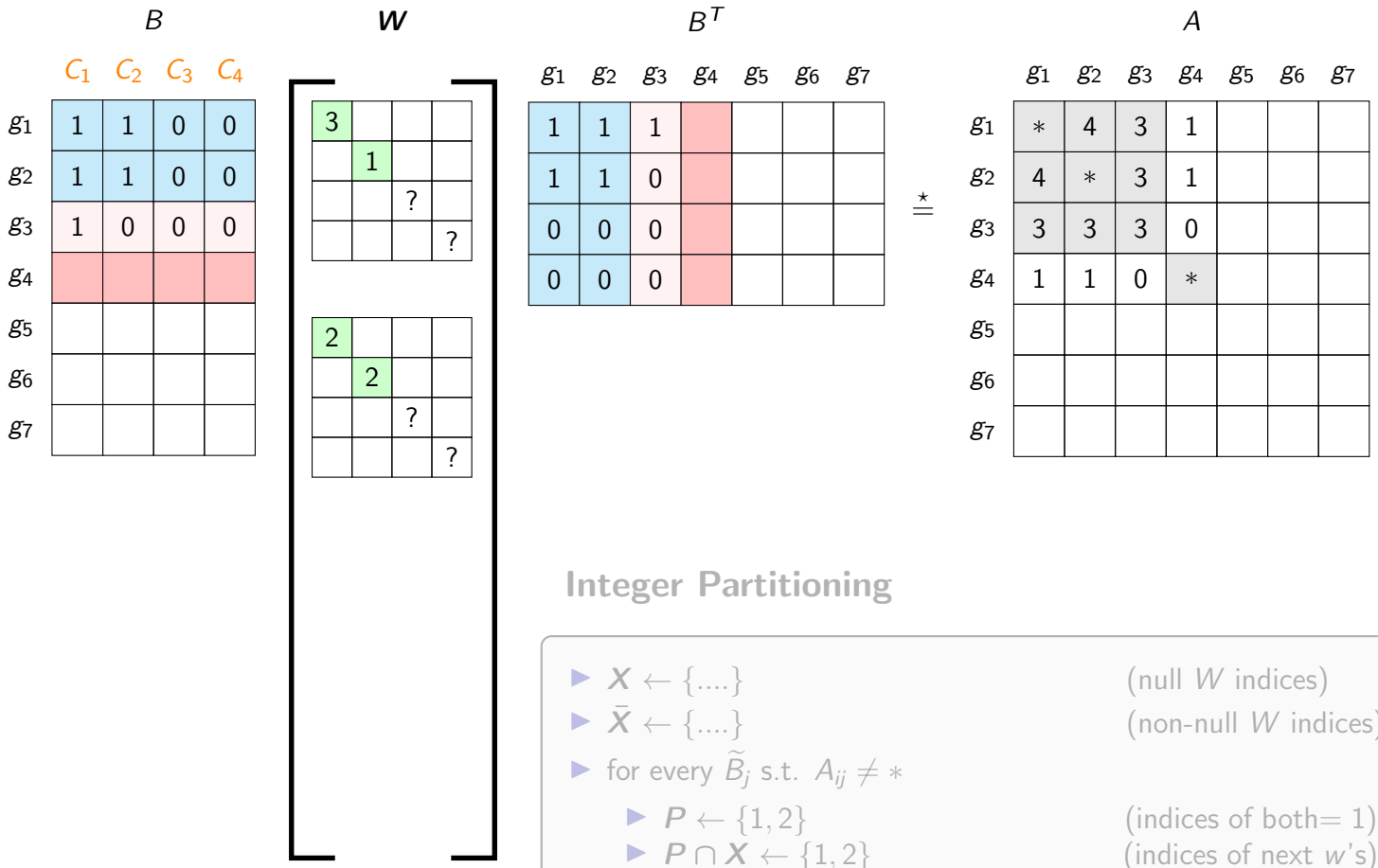


Integer Partitioning

- $X \leftarrow \{....\}$ (null W indices)
- $\bar{X} \leftarrow \{....\}$ (non-null W indices)
- for every $\widetilde{B}_j$ s.t. $A_{ij} \neq *$
  - $P \leftarrow \{1, 2\}$ (indices of both$= 1$)
  - $P \cap X \leftarrow \{1, 2\}$ (indices of next w's)
  - $\{3, 1\}, \{2, 2\} \leftarrow$ partitions of $A_{ij} = 4$ with 2 integers

Integer Partitioning Weight Recovery

3. iteratively fill in non-basis rows

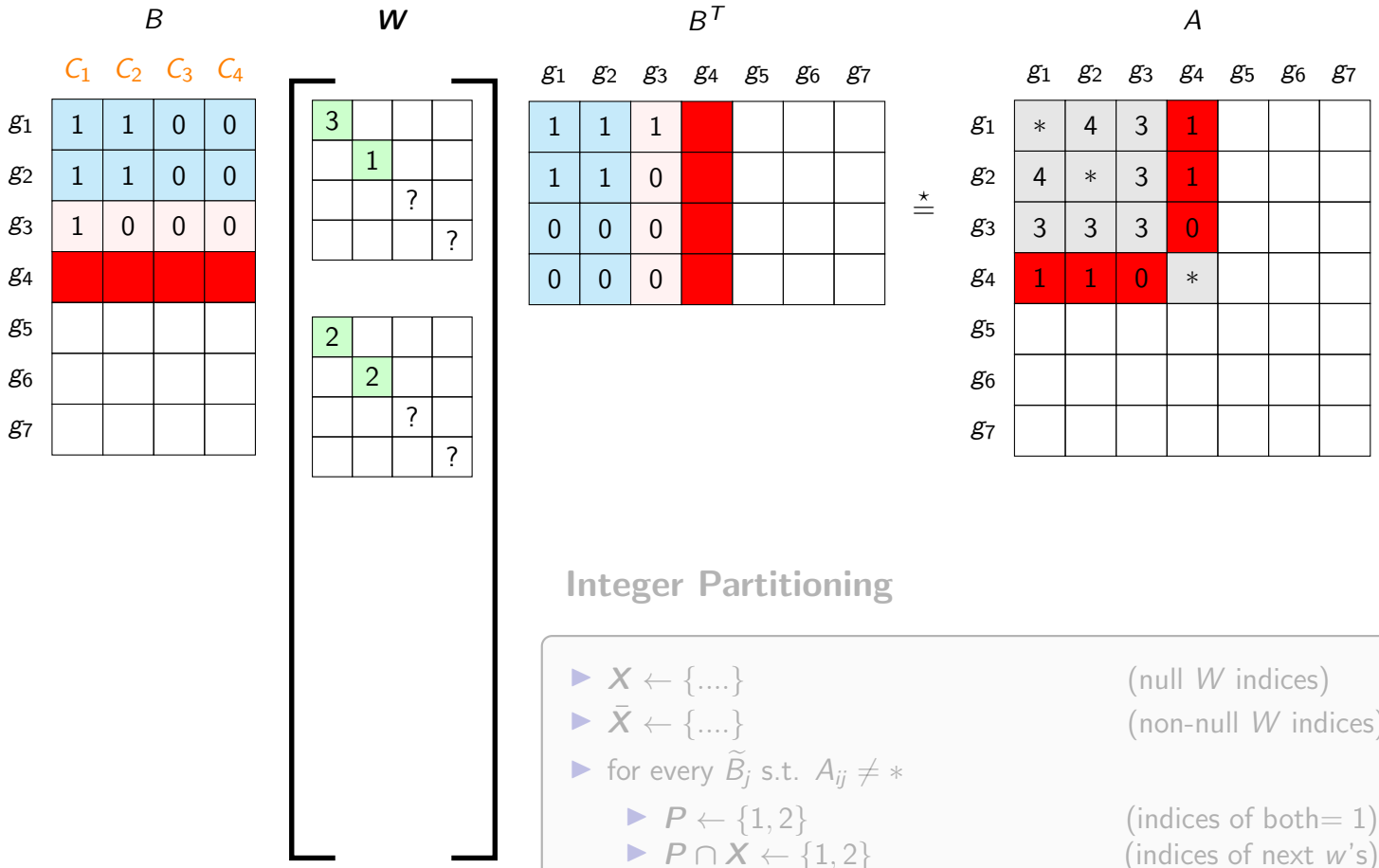


Integer Partitioning

- $X \leftarrow \{....\}$ (null W indices)
- $\bar{X} \leftarrow \{....\}$ (non-null W indices)
- for every $\widetilde{B}_j$ s.t. $A_{ij} \neq *$
  - $P \leftarrow \{1, 2\}$ (indices of both$= 1$)
  - $P \cap X \leftarrow \{1, 2\}$ (indices of next w's)
  - $\{3, 1\}, \{2, 2\} \leftarrow$ partitions of $A_{ij} = 4$ with 2 integers

Integer Partitioning Weight Recovery

3. iteratively fill in non-basis rows

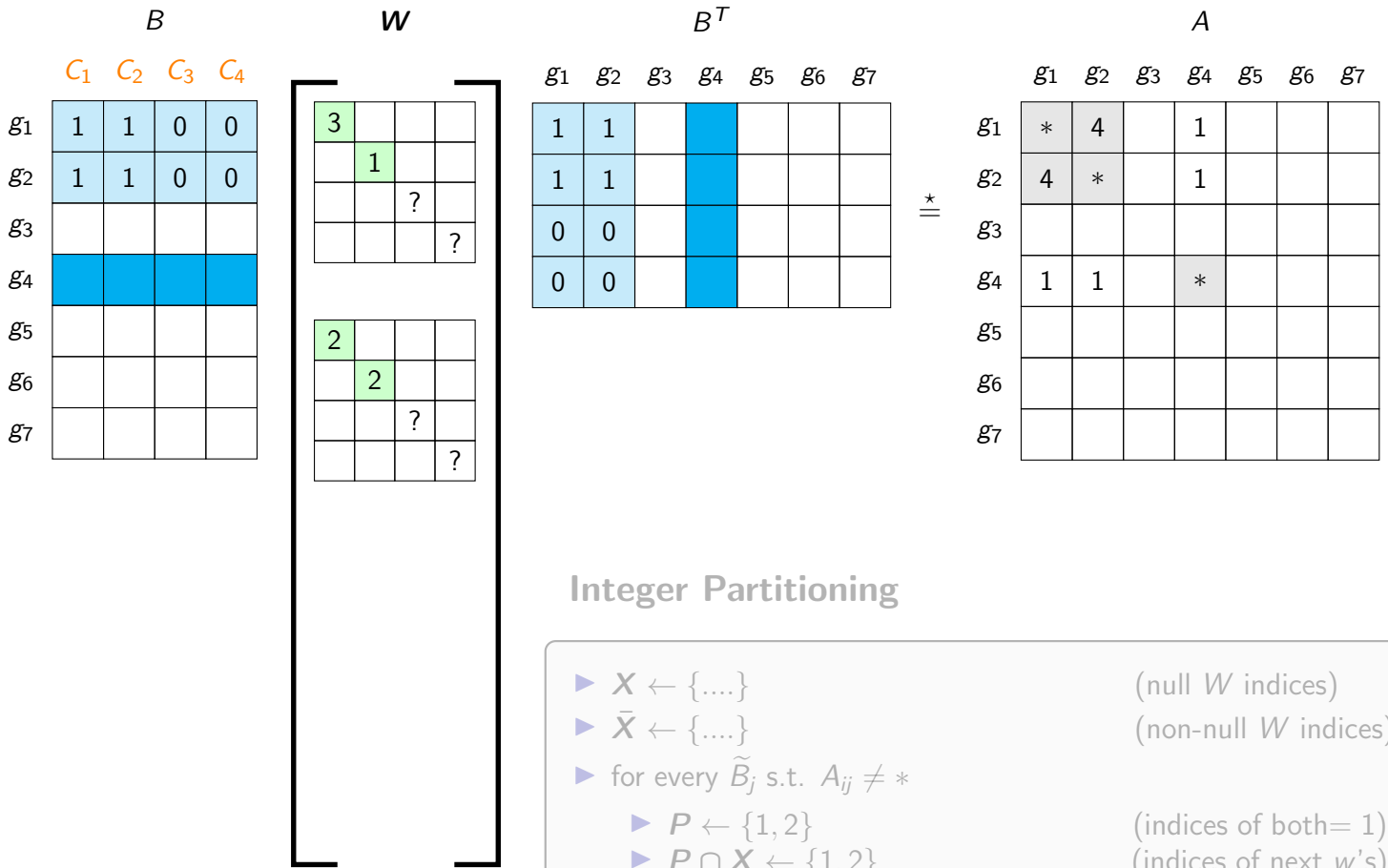


Integer Partitioning

- $X \leftarrow \{....\}$ (null W indices)
- $\bar{X} \leftarrow \{....\}$ (non-null W indices)
- for every $\widetilde{B}_j$ s.t. $A_{ij} \neq *$
  - $P \leftarrow \{1, 2\}$ (indices of both$= 1$)
  - $P \cap X \leftarrow \{1, 2\}$ (indices of next w's)
  - $\{3, 1\}, \{2, 2\} \leftarrow$ partitions of $A_{ij} = 4$ with 2 integers

Integer Partitioning Weight Recovery

1. guess basis row

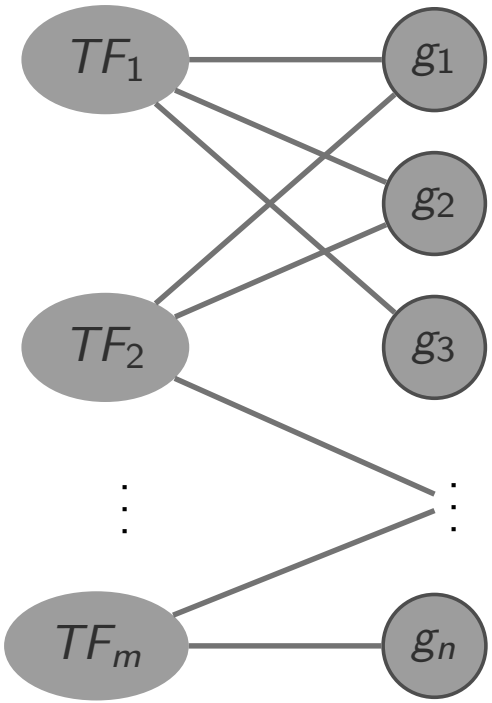


Integer Partitioning

- $X \leftarrow \{....\}$ (null W indices)
- $\bar{X} \leftarrow \{....\}$ (non-null W indices)
- for every $\widetilde{B}_j$ s.t. $A_{ij} \neq *$
  - $P \leftarrow \{1, 2\}$ (indices of both$= 1$)
  - $P \cap X \leftarrow \{1, 2\}$ (indices of next w's)
  - $\{3, 1\}, \{2, 2\} \leftarrow$ partitions of $A_{ij} = 4$ with 2 integers

Synthetic Graphs

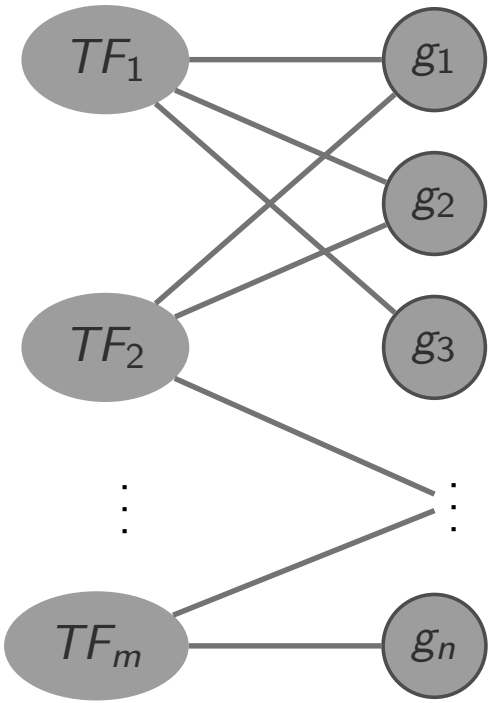
1. Transcription Factors



[3] From ML analysis (Taroni et al. 2019)

Synthetic Graphs

1. Transcription Factors

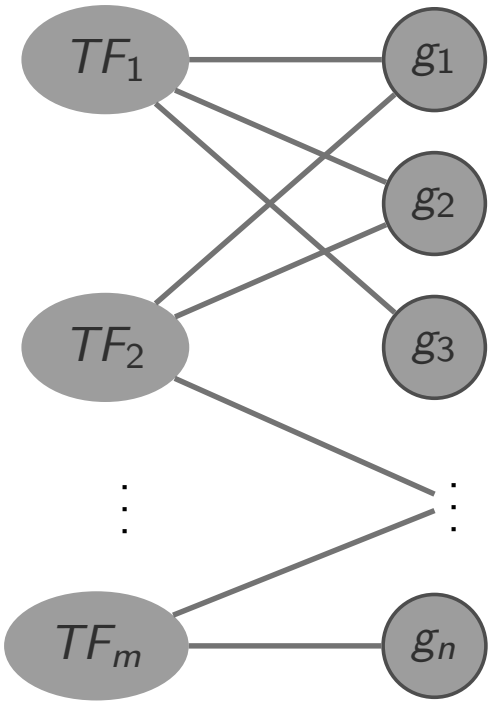


- Select k random modules for $k \in [2, 20]$ (20 seeds each)

[3] From ML analysis (Taroni et al. 2019)

Synthetic Graphs

1. Transcription Factors

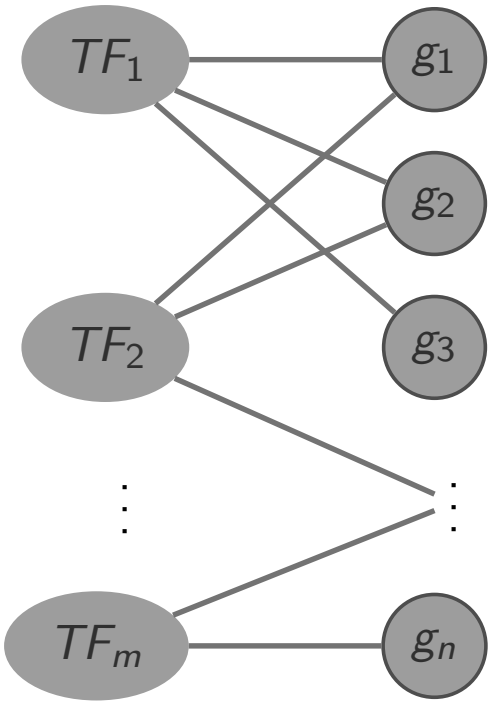


- Random, heavy-tailed clique weights
- Select k random modules for $k \in [2, 20]$ (20 seeds each)

[3] From ML analysis (Taroni et al. 2019)

Synthetic Graphs

1. Transcription Factors



2. Latent Variables[3]

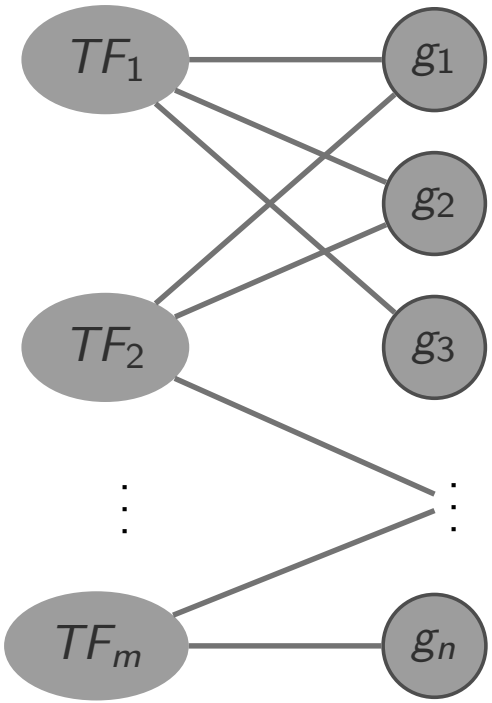
	lv_1	lv_2	$\cdots$	lv_n
g_1	0.99	0.85		0.20
g_2	0.08	0.85		0.10
$\vdots$				
g_m	0.95	0.40		0.65

- Random, heavy-tailed clique weights
- Select k random modules for $k \in [2, 20]$ (20 seeds each)

[3] From ML analysis (Taroni et al. 2019)

Synthetic Graphs

1. Transcription Factors



2. Latent Variables[3]

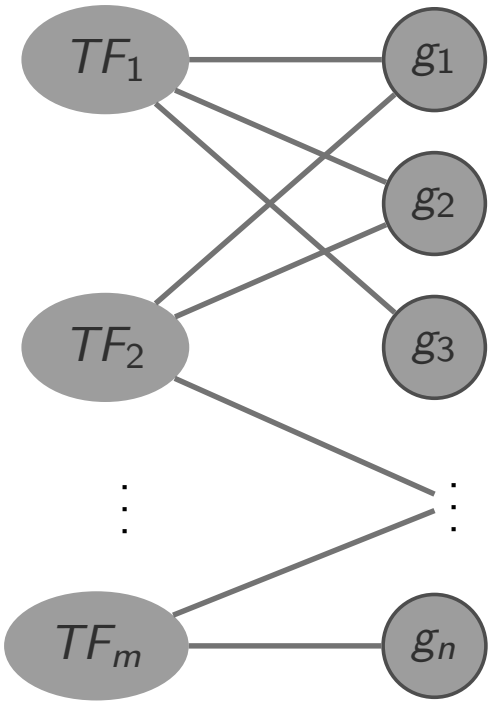
	lv_1	lv_2	$\cdots$	lv_n
g_1	0.99	0.85		0.20
g_2	0.08	0.85		0.10
$\vdots$				
g_m	0.95	0.40		0.65

- Random, heavy-tailed clique weights
- Select k random modules for $k \in [2, 20]$ (20 seeds each)

[3] From ML analysis (Taroni et al. 2019)

Synthetic Graphs

1. Transcription Factors



2. Latent Variables[3]

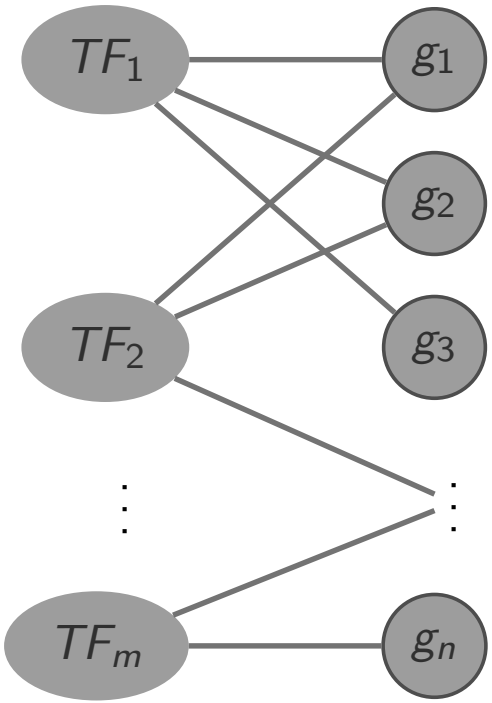
	lv_1	lv_2	$\cdots$	lv_n
g_1	0.99	0.85		
g_2		0.85		
$\vdots$				
g_m	0.95			0.65

- Random, heavy-tailed clique weights
- Select k random modules for $k \in [2, 20]$ (20 seeds each)

[3] From ML analysis (Taroni et al. 2019)

Synthetic Graphs

1. Transcription Factors



2. Latent Variables[3]

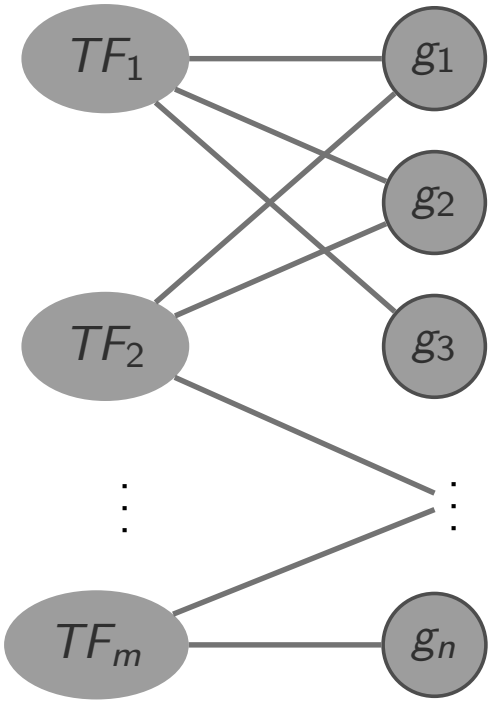
	lv_1	lv_2	$\cdots$	lv_n
g_1	0.99	0.85		
g_2		0.85		
$\vdots$				
g_m	0.95			0.65

- Random, heavy-tailed clique weights
- Transformed average for clique weights
- Select k random modules for $k \in [2, 20]$ (20 seeds each)

[3] From ML analysis (Taroni et al. 2019)

Synthetic Graphs

1. Transcription Factors



2. Latent Variables[3]

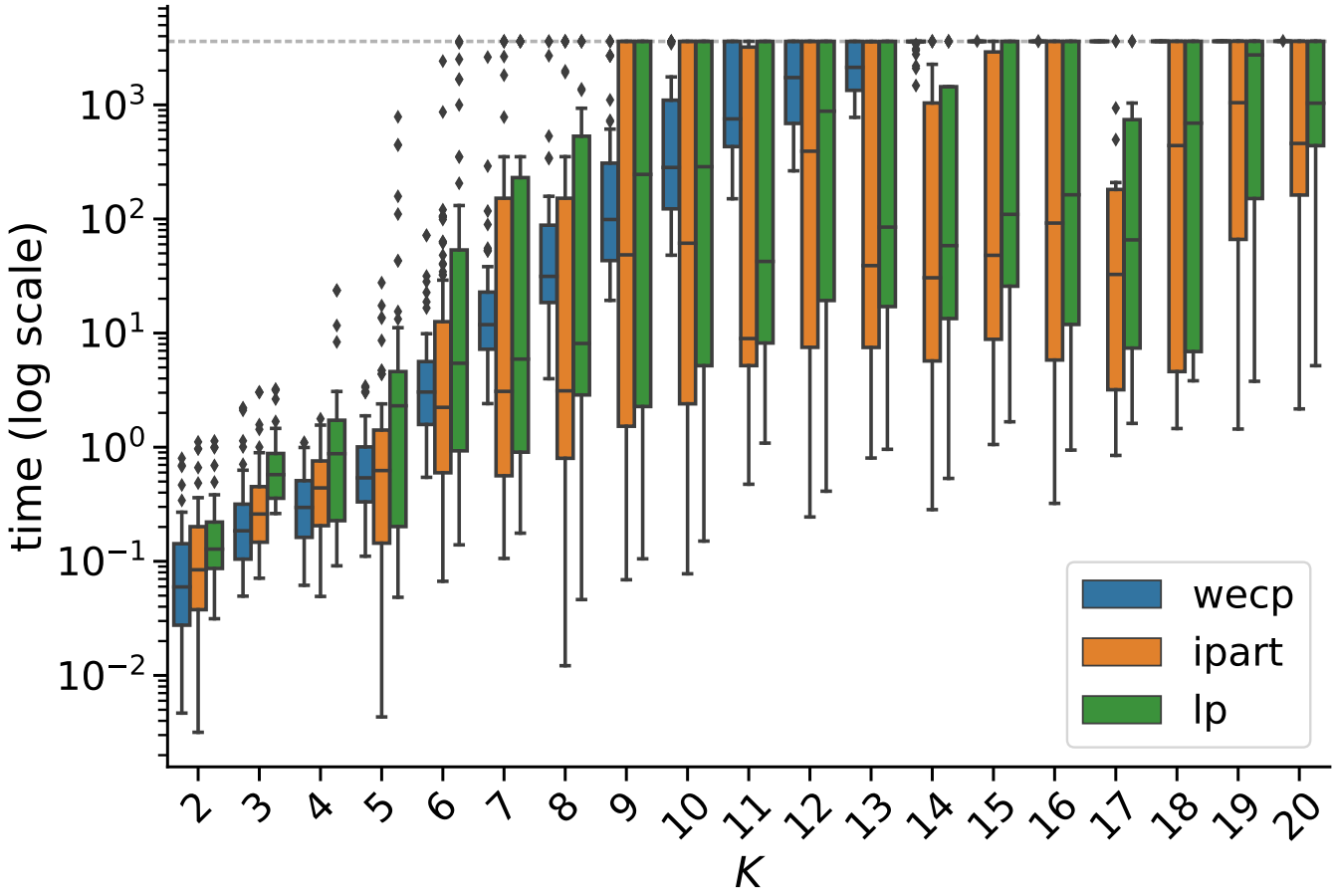
	lv_1	lv_2	$\cdots$	lv_n
g_1	0.99	0.85		
g_2		0.85		
$\vdots$				
g_m	0.95			0.65

- Random, heavy-tailed clique weights
- Transformed average for clique weights

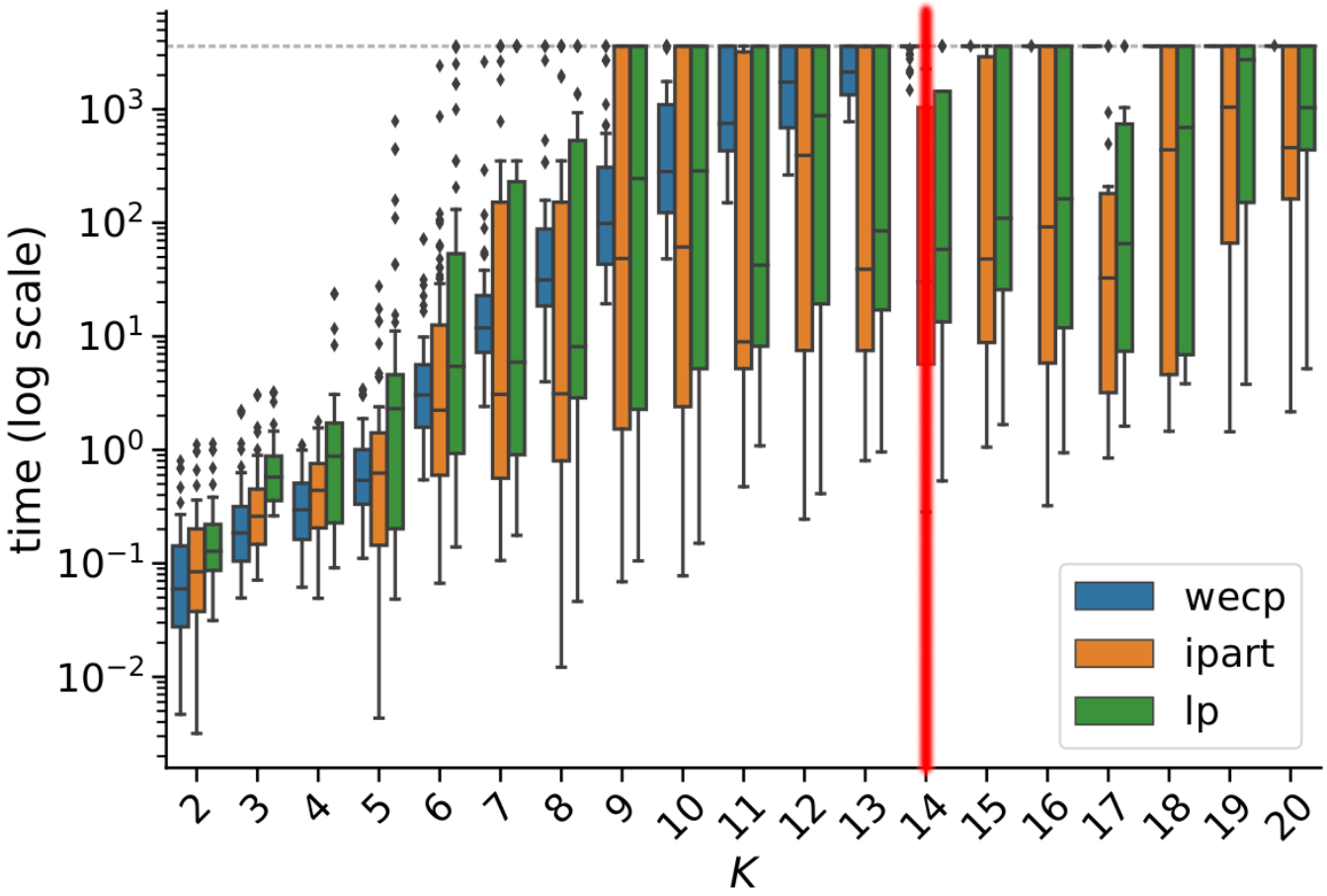
- Select k random modules for $k \in [2, 20]$ (20 seeds each)
- Graphs with $k \in [2, 11]$ after preprocessing

[3] From ML analysis (Taroni et al. 2019)

Results

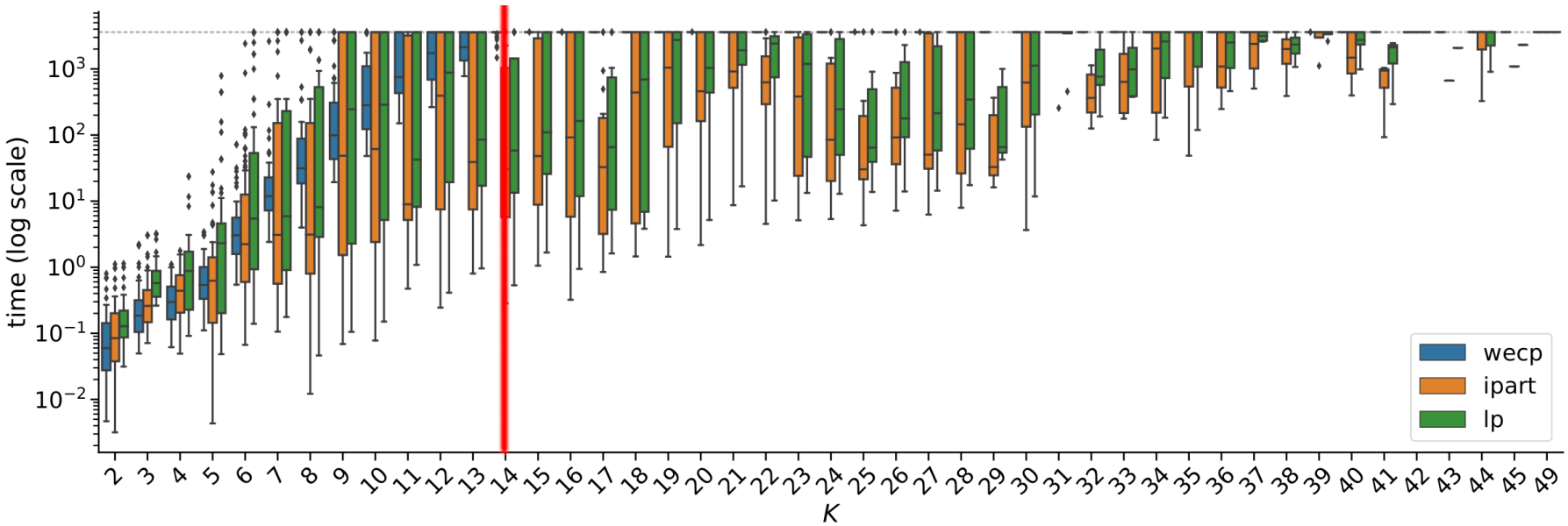

time (log scale)
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
K
wecp
ipart
lp

Results



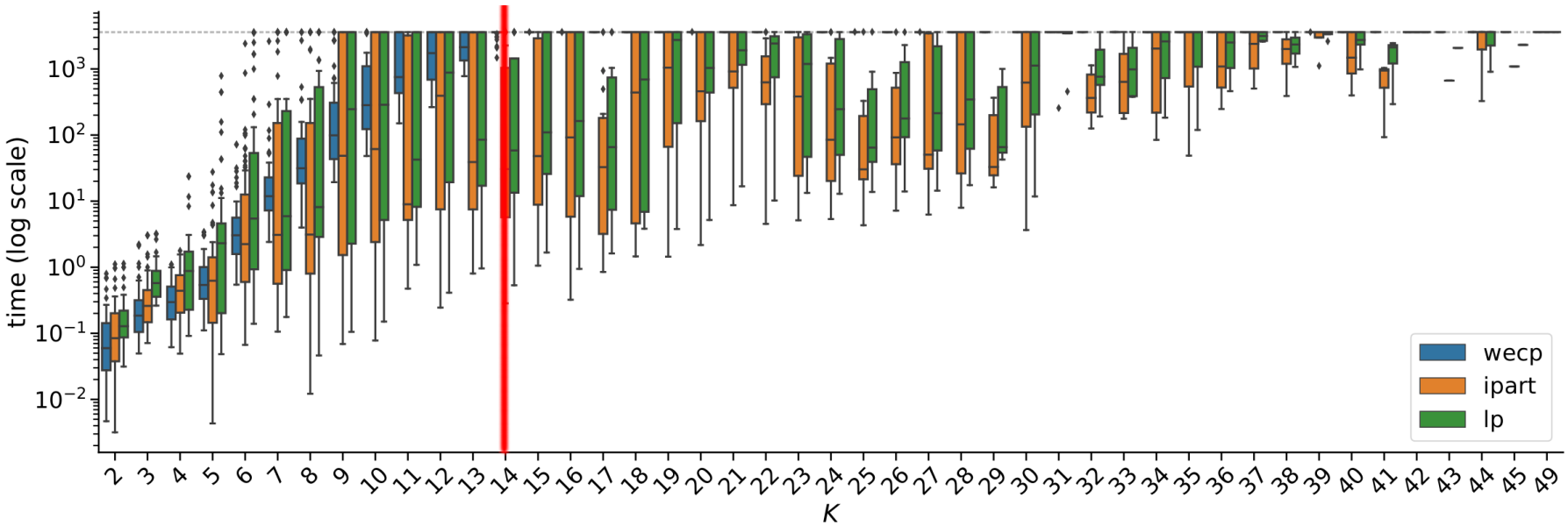
- wecp times out 100% of the time for $K > 14$

Results



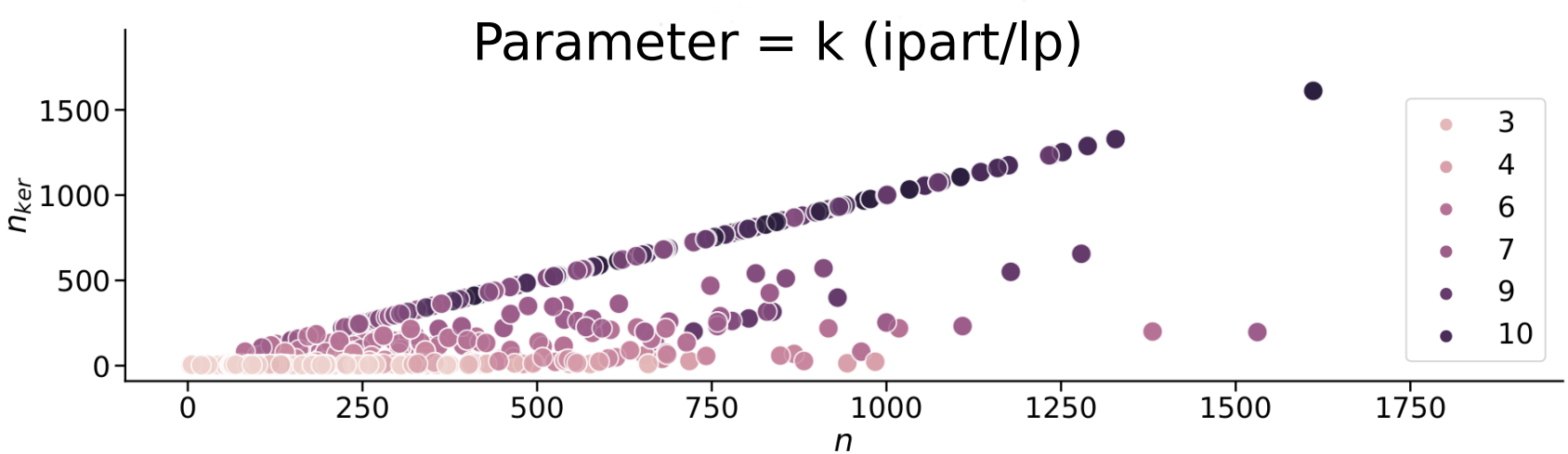
- wecp times out 100% of the time for $K > 14$

Results

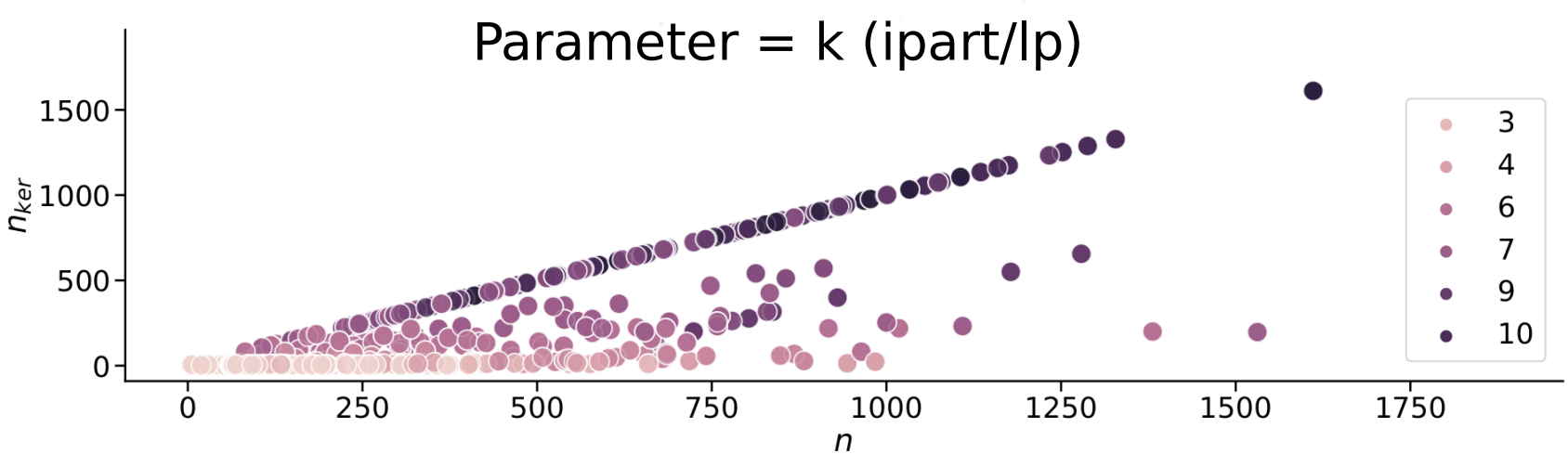


- wecp times out 100% of the time for $K > 14$
- ipart/lp methods able to compute solutions up to $K = 44$

Results

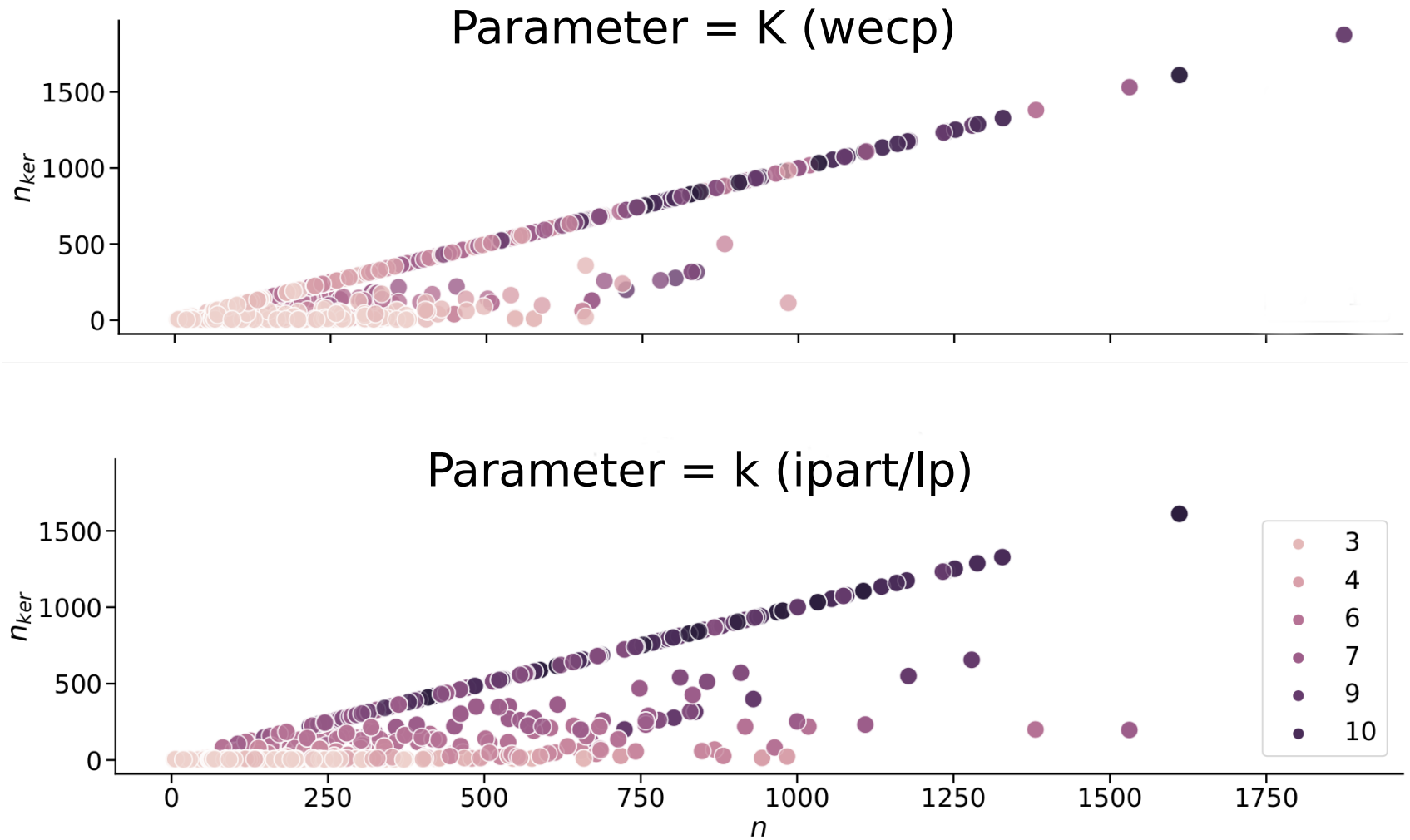

Parameter = k (ipart/lp)
n_{ker}
1500
1000
500
0
0
250
500
750
1000
1250
1500
1750
n
3
4
6
7
9
10

Results



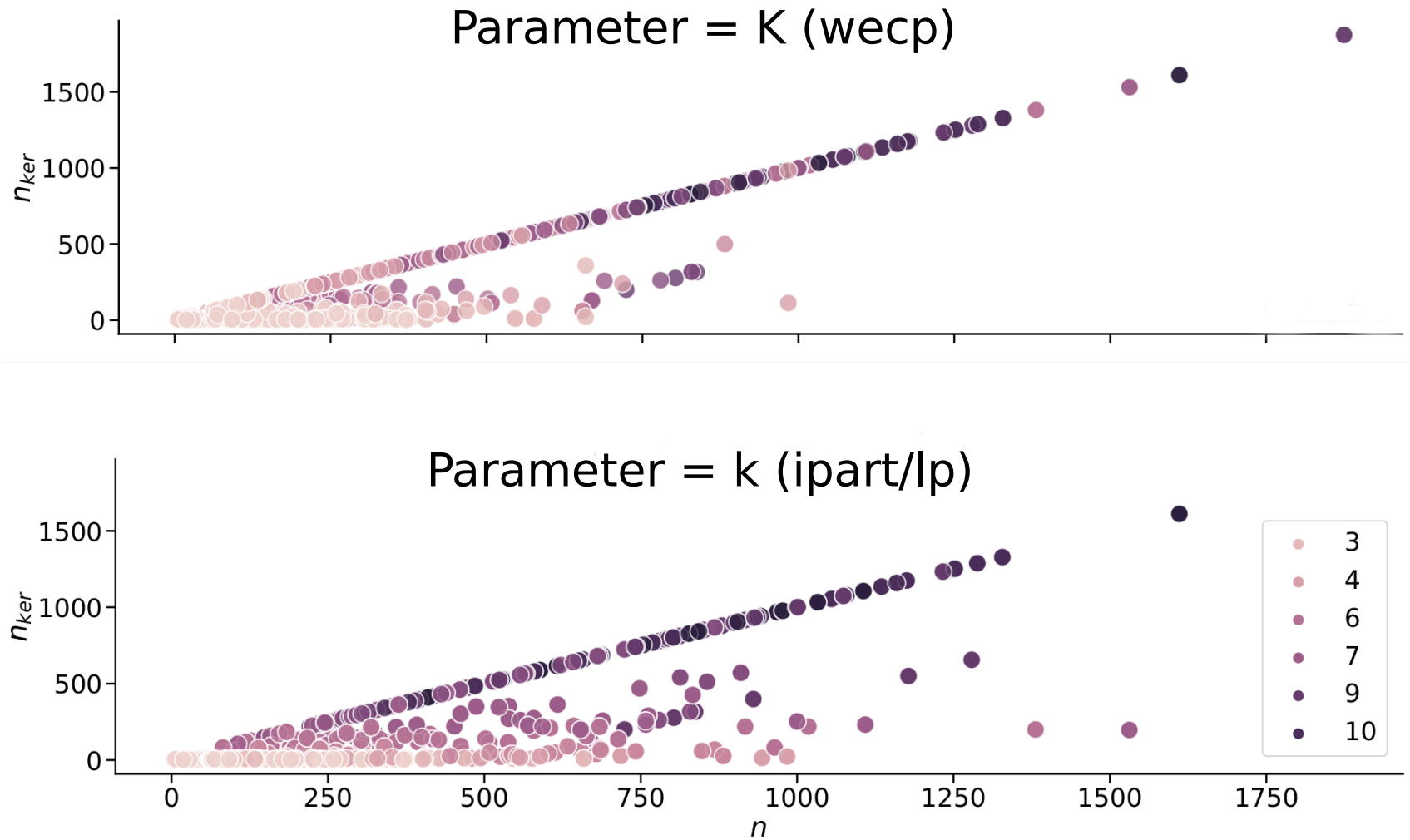
- (bottom) kernel gives more reduction for k parameter value

Results



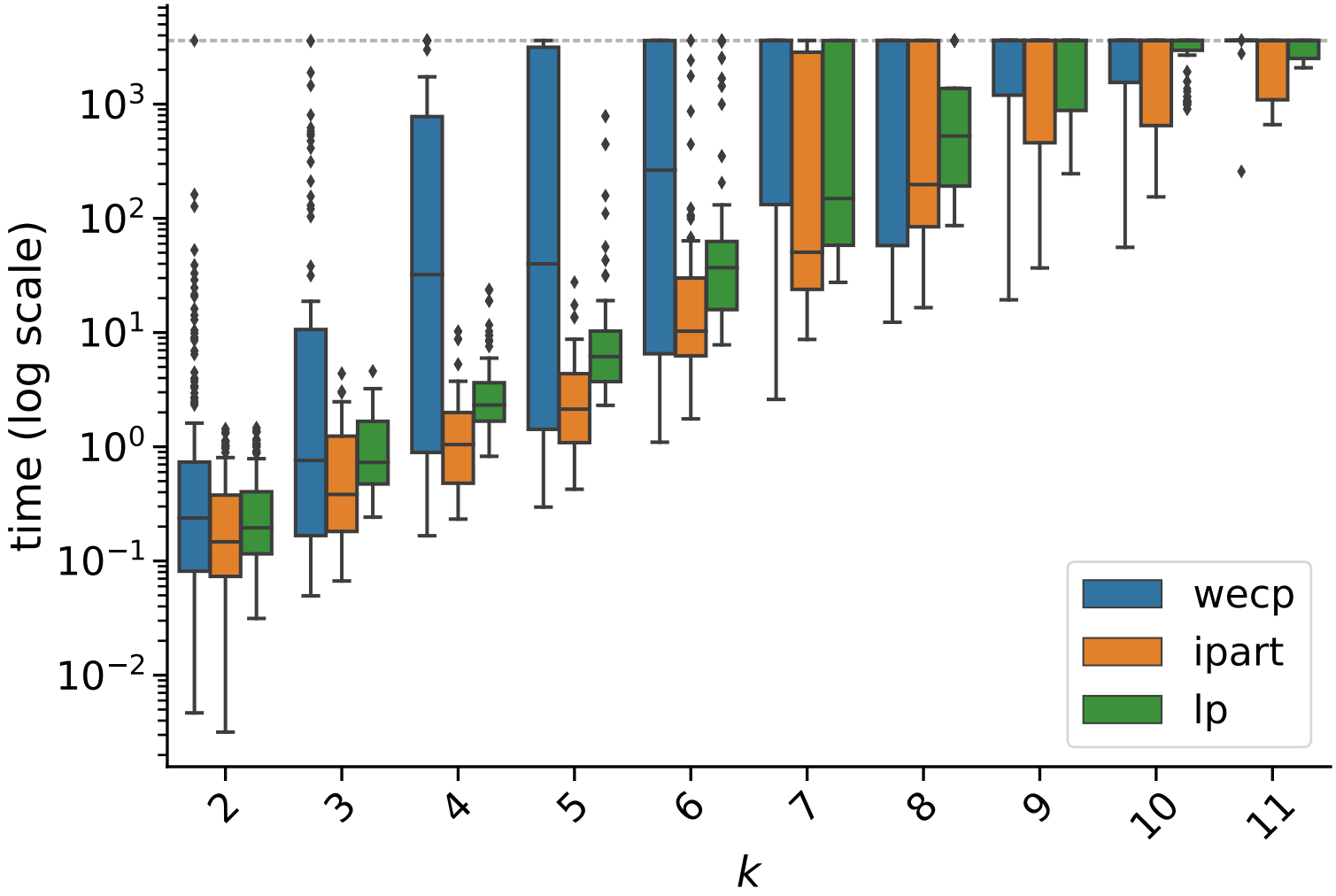
- (bottom) kernel gives more reduction for k parameter value

Results

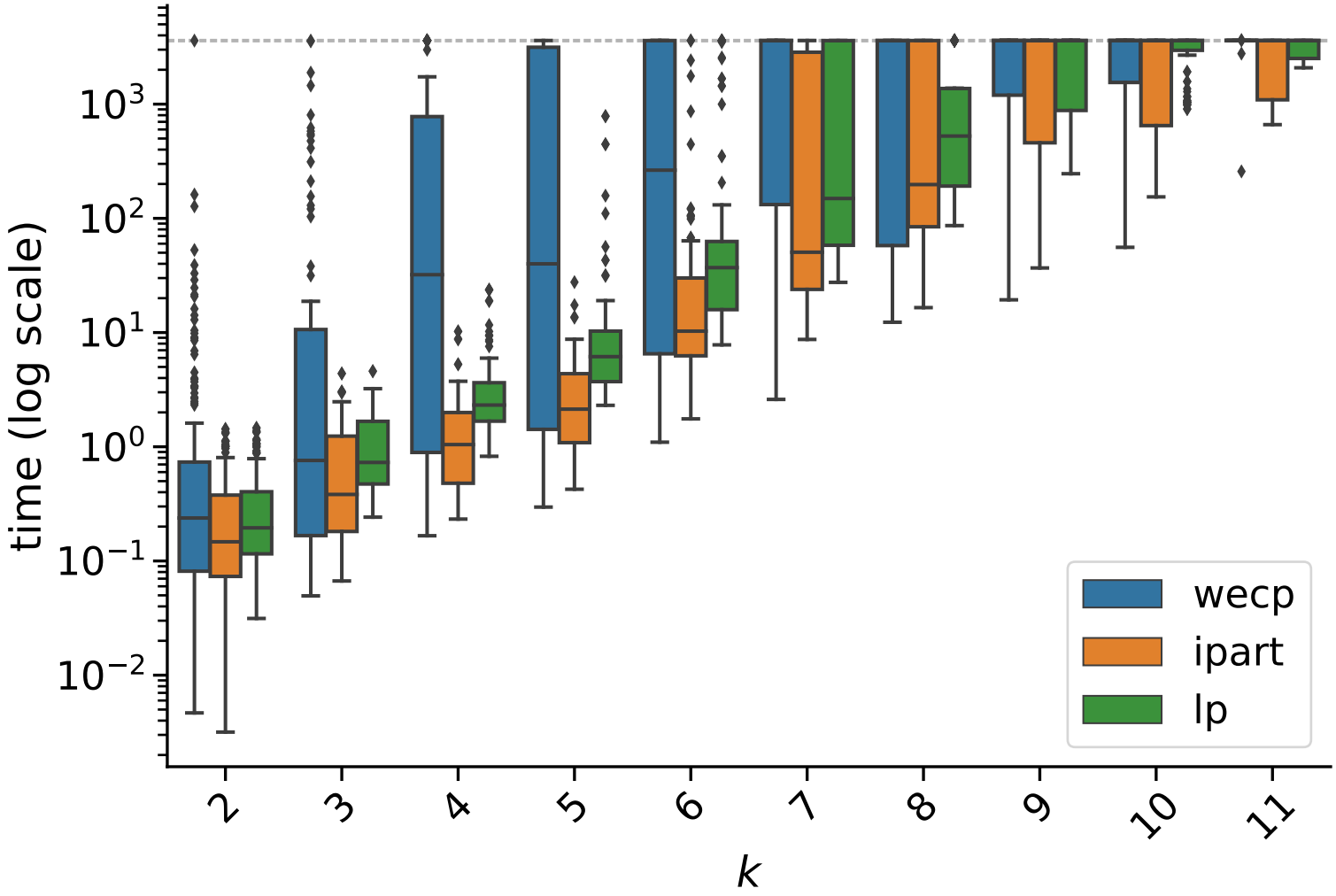


- (bottom) kernel gives more reduction for k parameter value
- (top) kernel gives less reduction when parameterized by K

Results


time (log scale)
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
2
3
4
5
6
7
8
9
10
11
k
wecp
ipart
lp

Results



- ipart/lp lower median runtimes for each k than wecp
- ipart slightly faster than lp

Future Work

- Noisy Setting
  - Penalty function for approximate edge weights
  - Hardness of problem depends on choice
- Hundreds of modules in real-world networks (k)
- New modeling decisions required

Thanks!

arXiv:2106.00657