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 | SIAM ACDA21

Biological Modules

Repair damage

Metabolism

Signal Transduction

Process drugs

Biological Modules

Repair damage

Metabolism

Signal Transduction

Process drugs

g_1

g_2

g_3

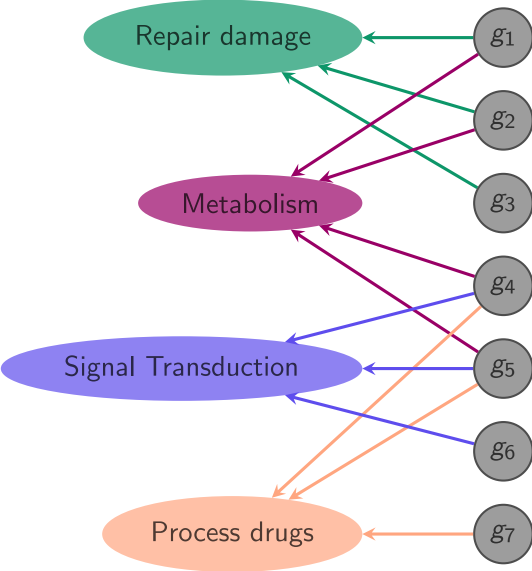
g_4

g_5

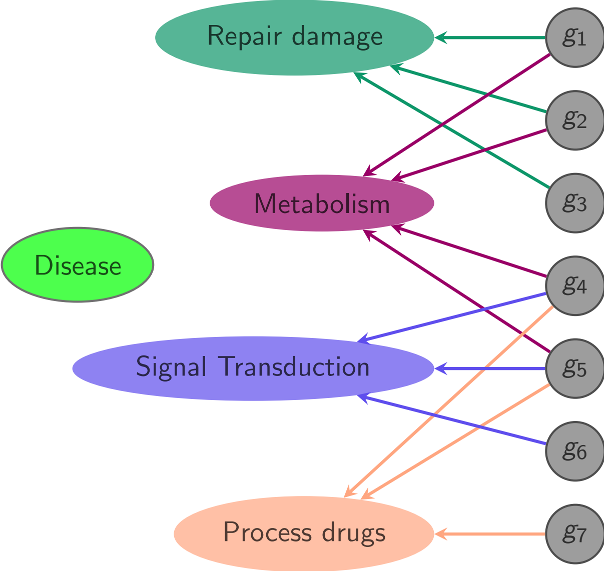
g_6

g_7

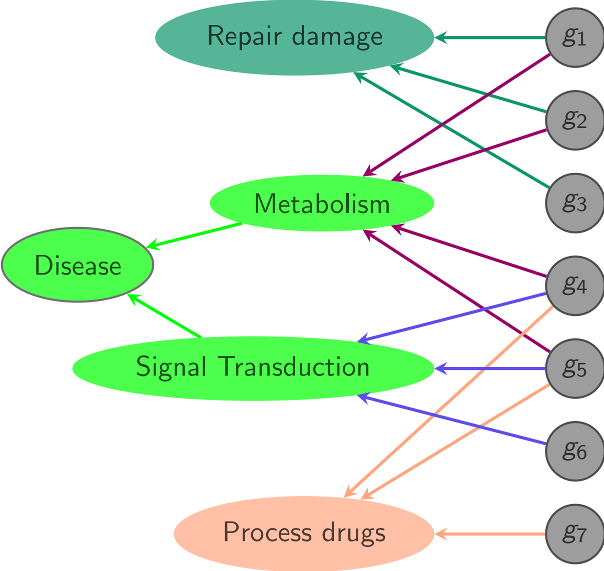
Biological Modules



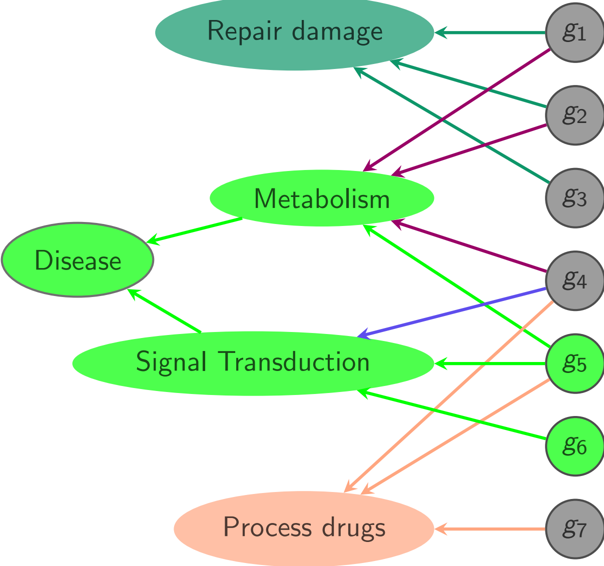
Biological Modules



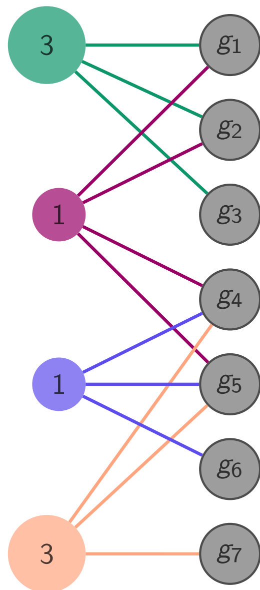
Biological Modules



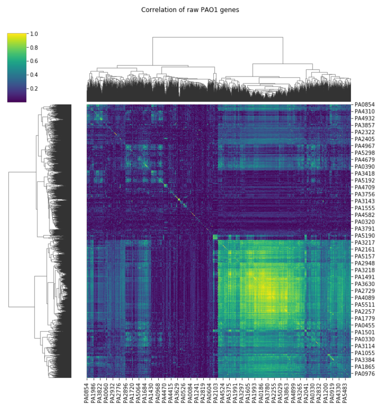
Biological Modules

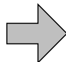


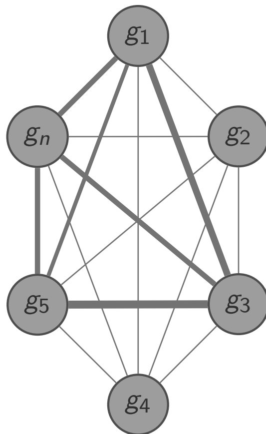
Problem Modeling



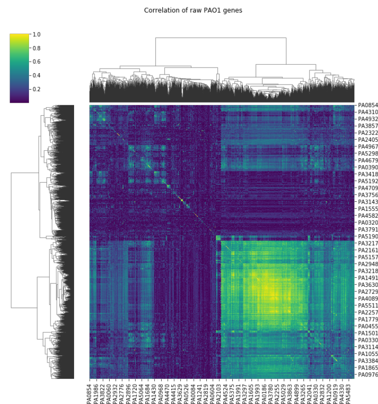
Gene Co-Expression Networks

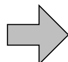


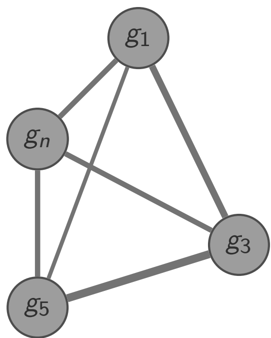




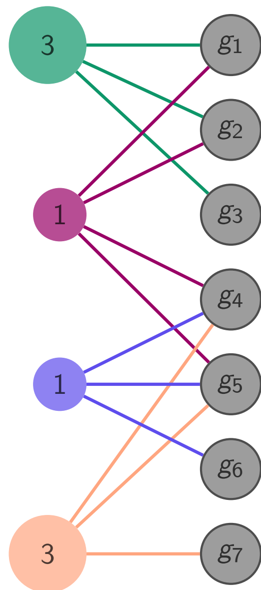
Gene Co-Expression Networks



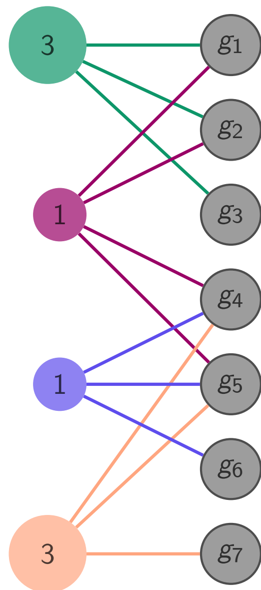


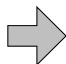


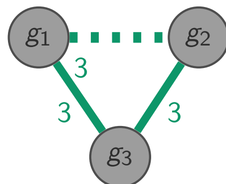
Gene-Gene Projection



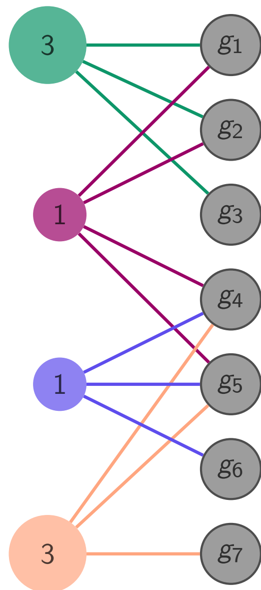
Gene-Gene Projection

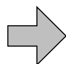


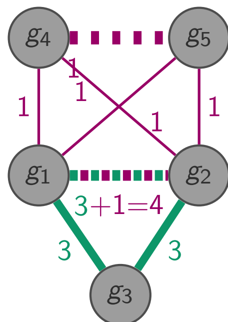




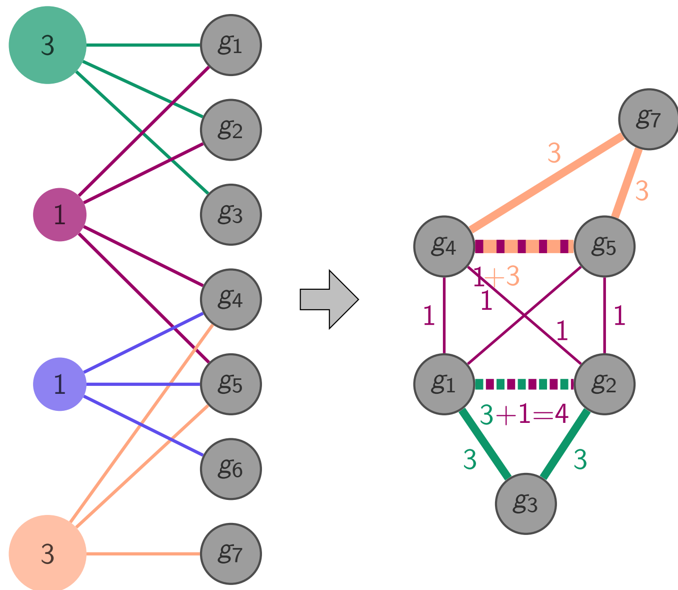
Gene-Gene Projection



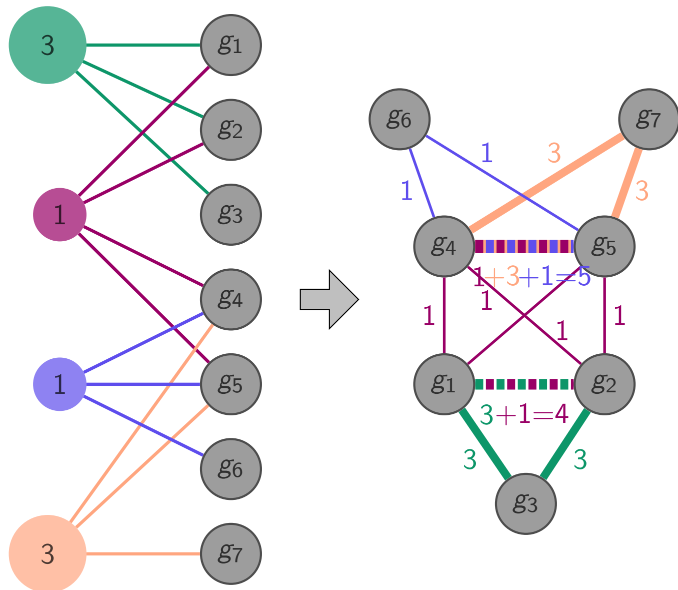




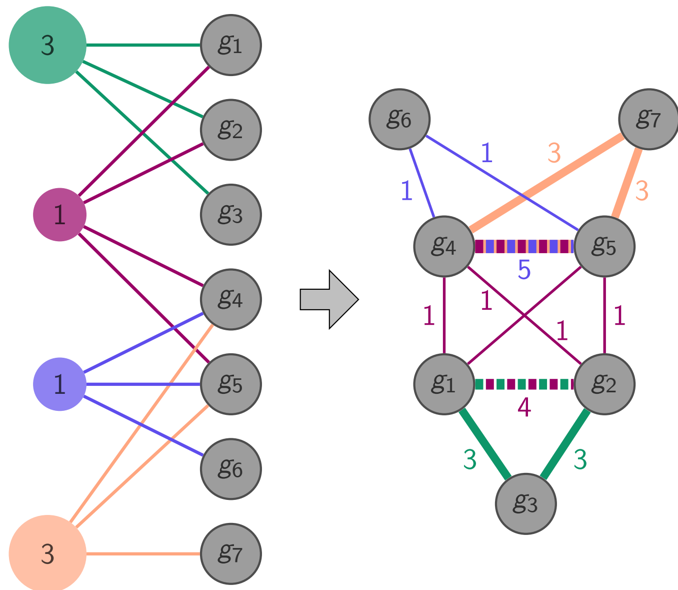
Gene-Gene Projection



Gene-Gene Projection



Challenges



Challenges

3 g_1

g_2

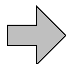
1 g_3

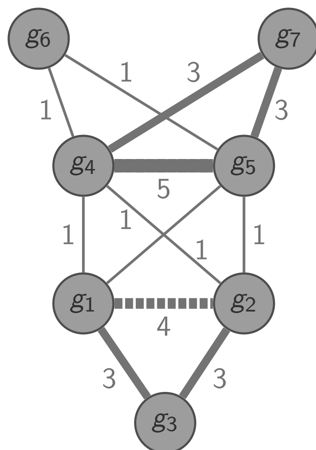
g_4

1 g_5

g_6

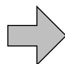
3 g_7

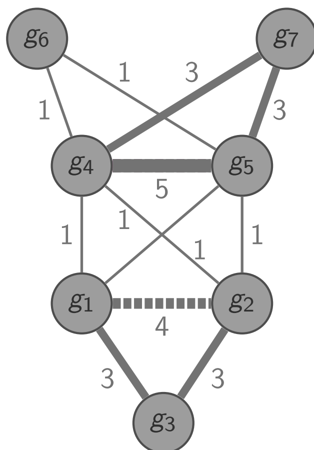




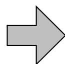
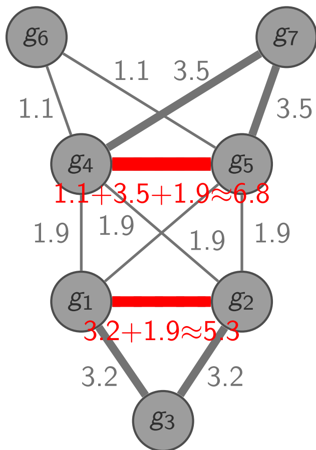
Challenges

⋮



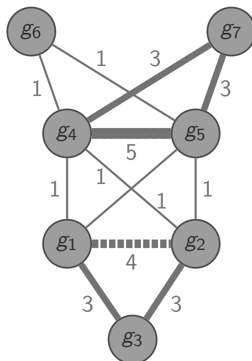


Challenges

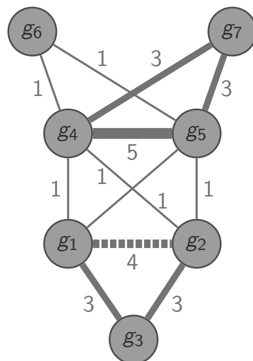
Restricted Setting

- 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[1].

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.

[1] 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[1].

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.

[1] with max-edge weight w

Prior Work

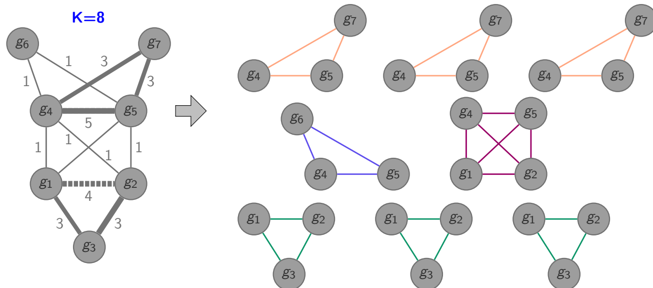
- Feldmann et al. (2020) give $2^{O(K^{3/2} w^{1/2} \log(K/w))} + O(n^2 \log n)$ FPT algorithm for similar problem[1].

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.

- Input parameter K represents *total # of cliques*.



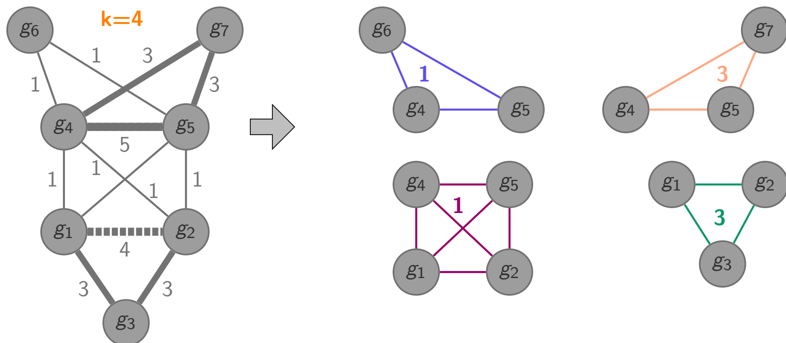
[1] with max-edge weight w

Contributions

- Exact Weighted Clique Decomposition

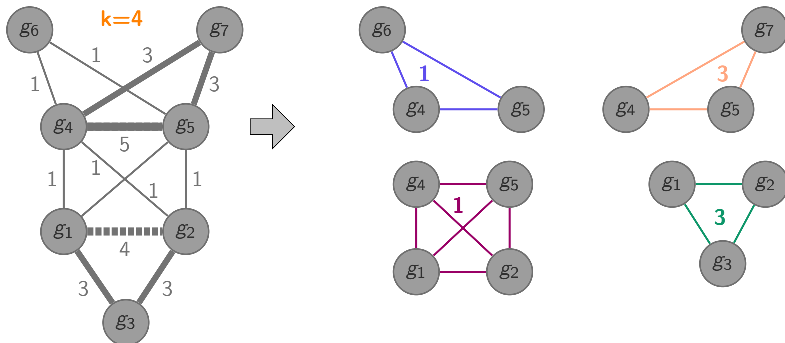
Contributions

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



Contributions

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



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

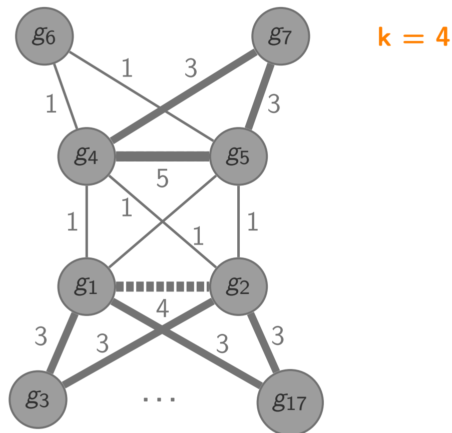
Pipeline

k = 8

Pipeline

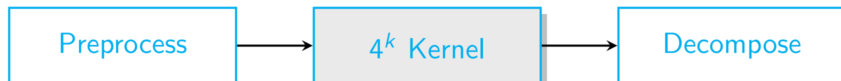
k = 8 -4

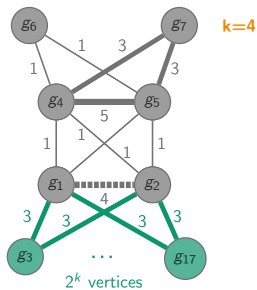
Pipeline



k = 4

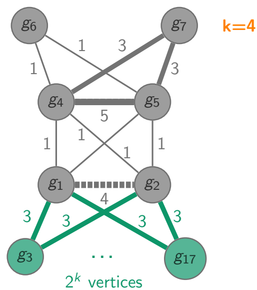
Pipeline

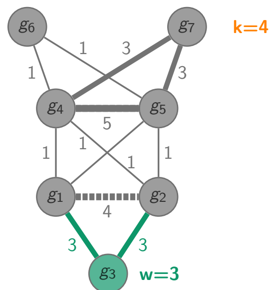




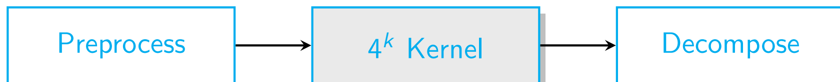
k=4

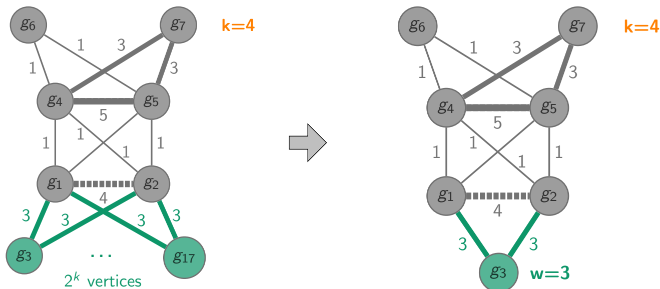
Pipeline





Pipeline



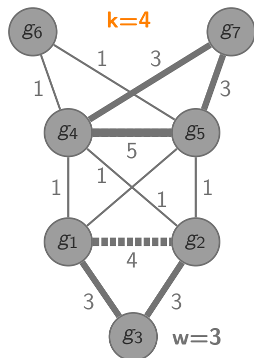


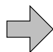
Annotated EWCD

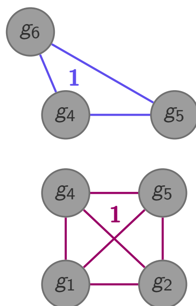
Input: a graph G, non-negative edge weights, set of vertex weights, integer k.

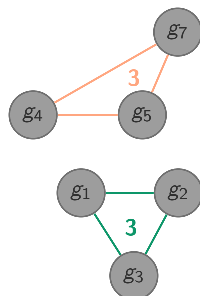
Output: a set of at most k weighted cliques such that the sum of containing cliques on each edge agrees with the edge weight.

Pipeline







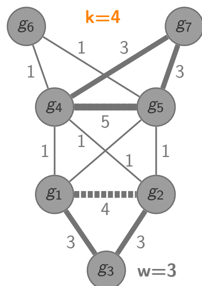


Matrix Reformulation

- Both algorithms solve equivalent matrix reformulations

Matrix Reformulation

- Both algorithms solve equivalent matrix reformulations

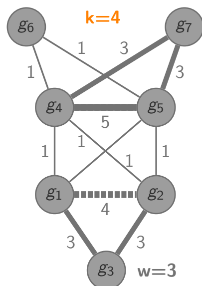


	g_1	g_2	g_3	g_4	g_5	g_6	g_7
g_1	*	4	3	1	1	0	0
g_2	4	*	3	1	1	0	0
g_3	3	3	3	0	0	0	0
g_4	1	1	0	*	5	1	3
g_5	1	1	0	5	*	1	3
g_6	0	0	0	1	1	*	0
g_7	0	0	0	3	3	0	*

$= A$

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

$\stackrel{*}{=}$

A

	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	*

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

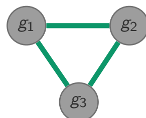
B^T

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

	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	*



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

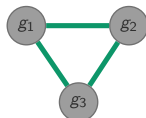
B^T

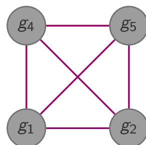
g_1	g_2	g_3	g_4	g_5	g_6	g_7
1	1	1	0	0	0	0
1	1	0	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	1

$\stackrel{*}{=}$

A

	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	*





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

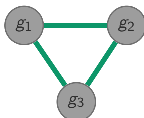
B^T

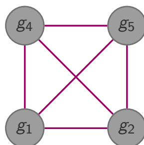
g_1	g_2	g_3	g_4	g_5	g_6	g_7
1	1	1	0	0	0	0
1	1	0	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	1

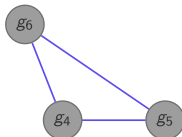
$\stackrel{*}{=}$

A

	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	*







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

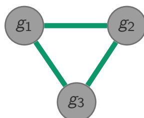
B^T

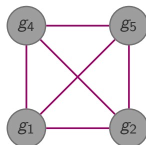
g_1	g_2	g_3	g_4	g_5	g_6	g_7
1	1	1	0	0	0	0
1	1	0	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	1

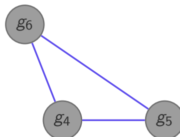
$\stackrel{\star}{=}$

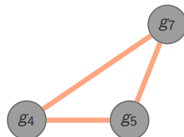
A

	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	*

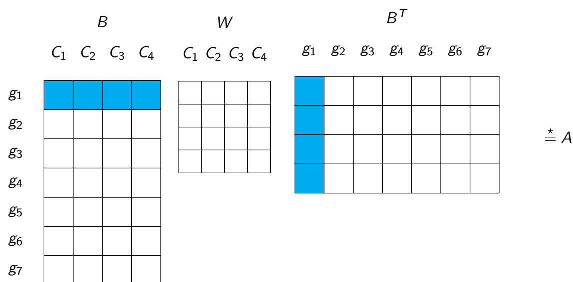






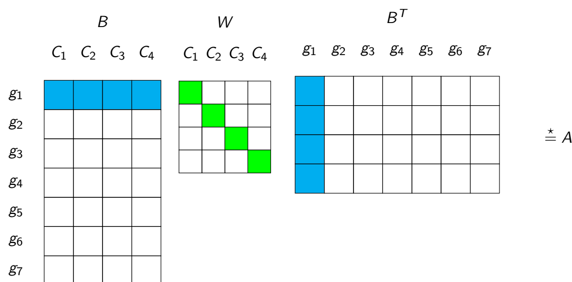


Algorithm Overview



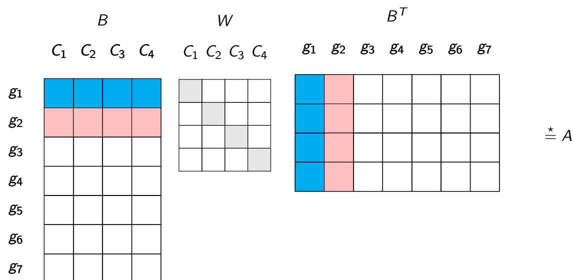
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Algorithm Overview



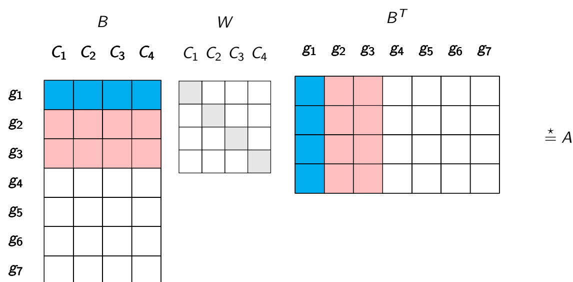
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Algorithm Overview



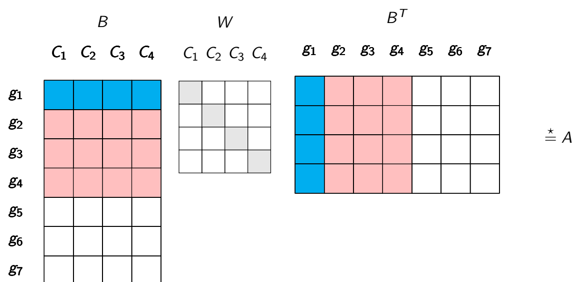
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Algorithm Overview



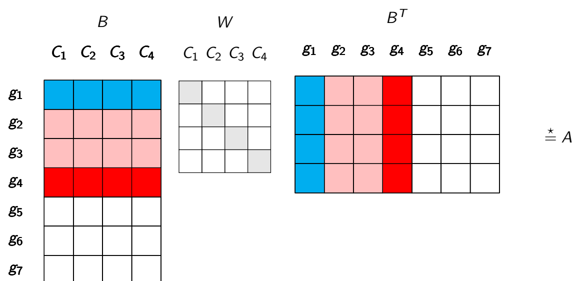
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Algorithm Overview



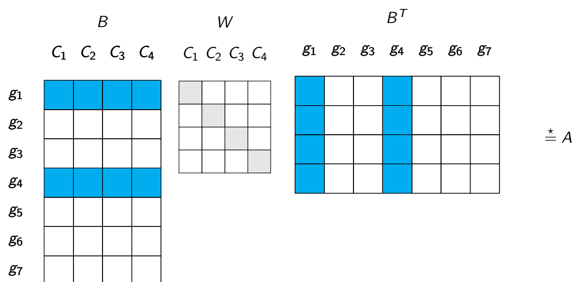
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Algorithm Overview



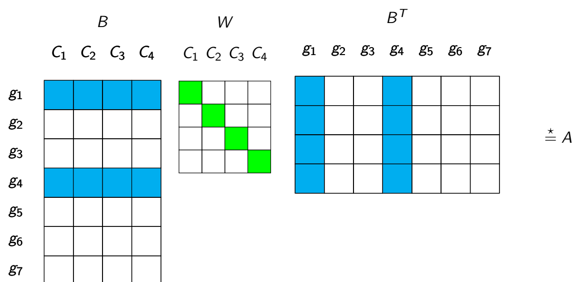
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Algorithm Overview



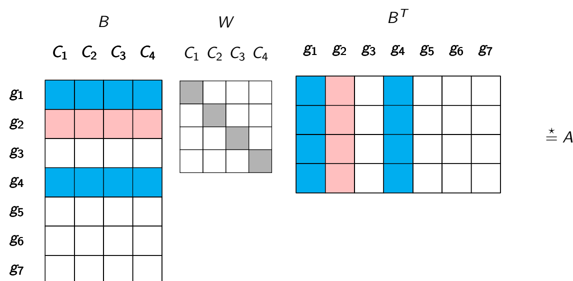
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

Algorithm Overview



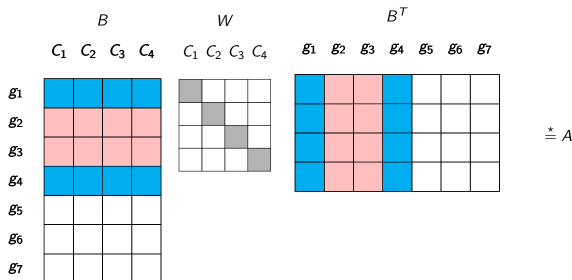
1. Guess basis row
2. Solve for clique weights using either *linear programming* or *integer partitioning* subroutines
3. Iteratively fill in non-basis rows

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

Clique Weight Recovery

1. Linear Programming (lp)
   - EWCD can be solved in $O(4^{k^2}k^3(32^k + k^3L))$ [L gives # of bits]
   - May work with non-integral edge weights

Clique Weight Recovery

1. Linear Programming (lp)
   - EWCD can be solved in $O(4^{k^2}k^3(32^k + k^3L))$ [L gives # of bits]
   - May work with non-integral edge weights
2. Integer Partitioning (ipart)
   - EWCD can be solve in $O(4^{k^2}32^kM^kk)$ when M is integral [M gives max edge-weight]
   - Won't work with non-integral edge weights

Synthetic Graphs

Used two regulatory module proxies

1. Transcription Factors (TFs)
   - ground-truth gene-TF associations
   - random, heavy-tailed clique weights

Synthetic Graphs

Used two regulatory module proxies

1. Transcription Factors (TFs)
   - ground-truth gene-TF associations
   - random, heavy-tailed clique weights
2. Latent Variables (LVs)
   - from ML analysis (Taroni et al. 2019)
   - threshold strength of association to select genes
   - cliques weights taken from transformed average

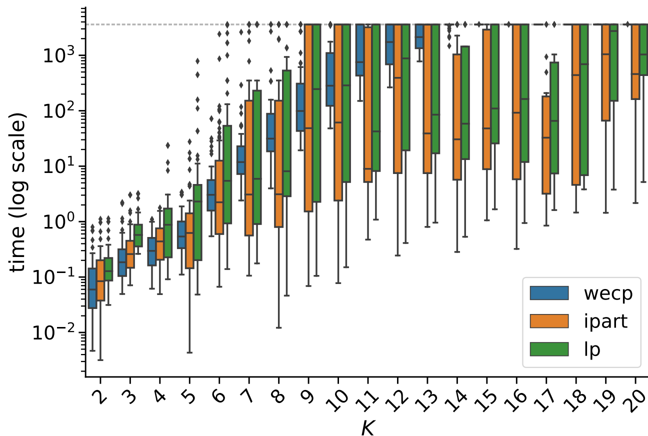
Synthetic Graphs

Used two regulatory module proxies

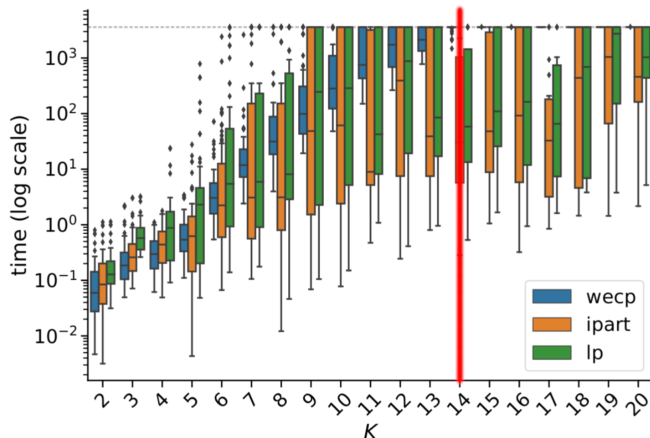
1. Transcription Factors (TFs)
   - ground-truth gene-TF associations
   - random, heavy-tailed clique weights
2. Latent Variables (LVs)
   - from ML analysis (Taroni et al. 2019)
   - threshold strength of association to select genes
   - cliques weights taken from transformed average

- k values initially ranged from $2 - 20$
- k values $2 - 11$ after pre-processing
- 20 random seeds

Results

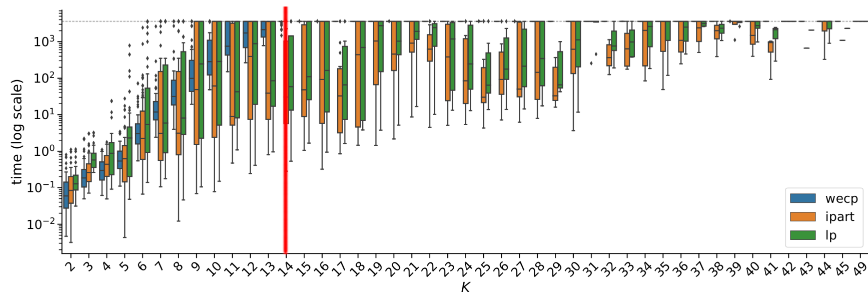


Results



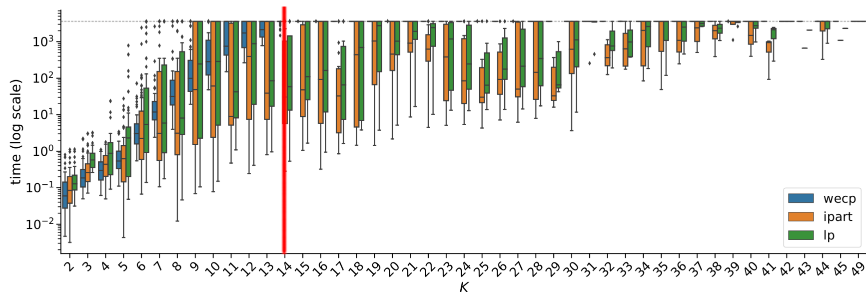
- wecp times out 100% of the time for $K > 14$

Results



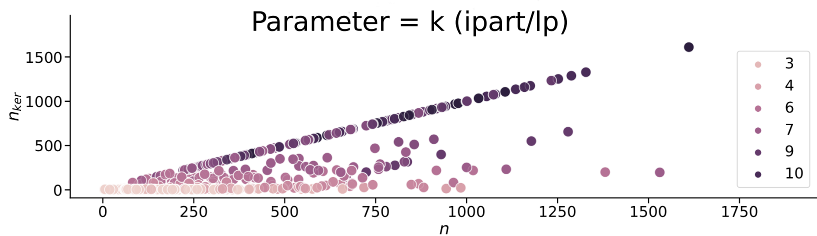
- wecp times out 100% of the time for $K > 14$

Results

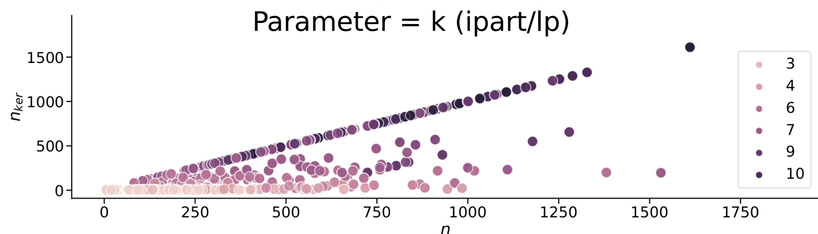


- wecp times out 100% of the time for $K > 14$
- ipart/lp methods able to compute solutions up to $K = 44$

Results

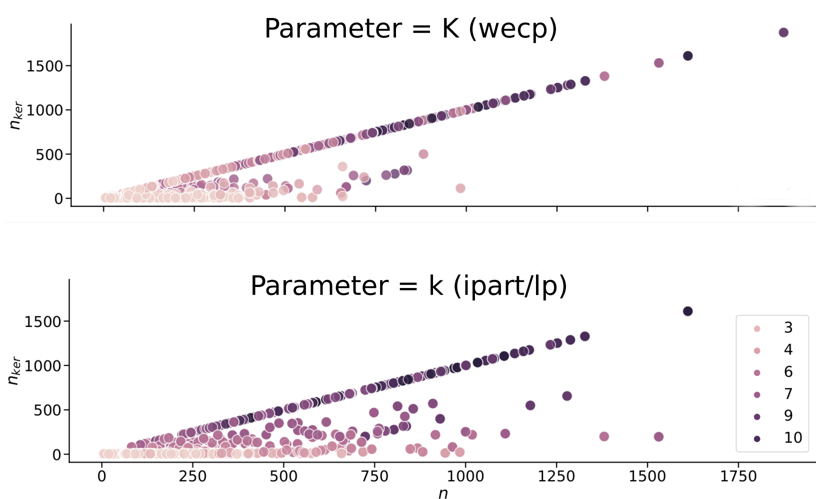


Results



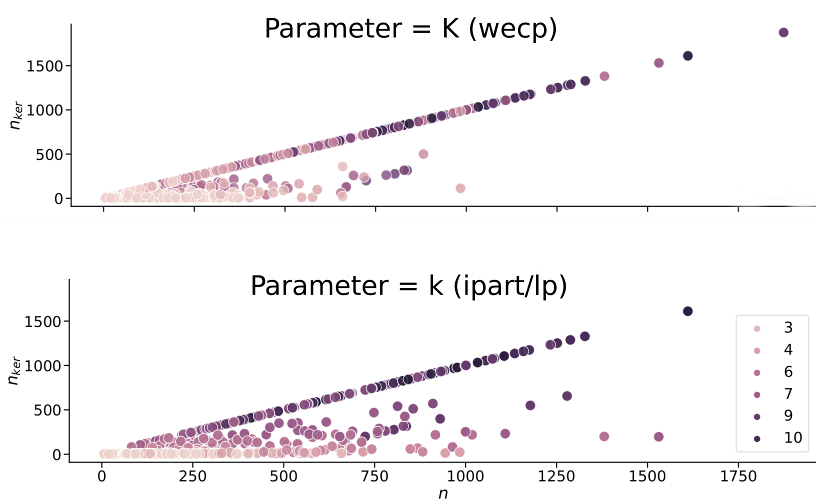
- (bottom) kernel gives more reduction for k parameter value

Results



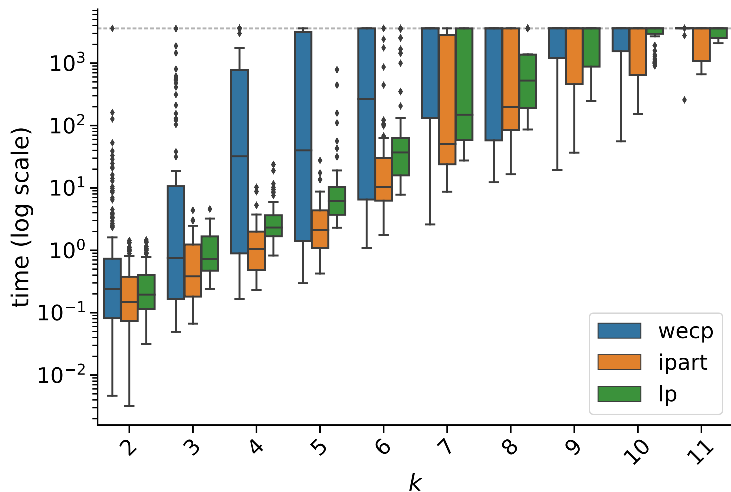
- (bottom) kernel gives more reduction for k parameter value

Results

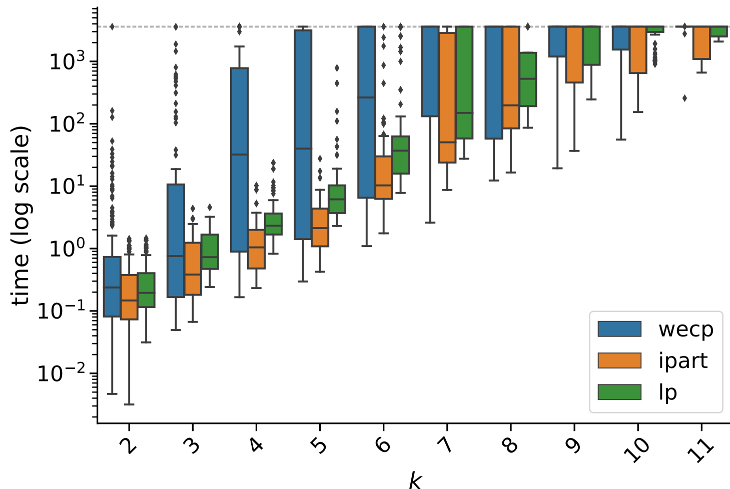


- (bottom) kernel gives more reduction for k parameter value
- (top) kernel gives less reduction when parameterized by K

Results



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!

Lightning talk: Tuesday, July 20th at 4:55pm EDT

[2]

GORDON AND BETTY MOORE FOUNDATION

[2] Photo of Gene Simmons from metalheadzone.com