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

Madison Cooley[†] Casey S. Greene[‡] Davis Issac[§] Milton Pividori[¶]
Blair D. Sullivan[‖]

September 8, 2021

arXiv:2106.00657v2 [cs.DS] 7 Sep 2021

Abstract

We present a new combinatorial model for identifying regulatory modules in gene co-expression data using a decomposition into weighted cliques. To capture complex interaction effects, we generalize the previously-studied weighted edge clique partition problem. As a first step, we restrict ourselves to the noise-free setting, and show that the problem is fixed parameter tractable when parameterized by the number of modules (cliques). We present two new algorithms for finding these decompositions, using linear programming and integer partitioning to determine the clique weights. Further, we implement these algorithms in Python and test them on a biologically-inspired synthetic corpus generated using real-world data from transcription factors and a latent variable analysis of co-expression in varying cell types.

1 Introduction

Biomedical research has recently seen a burgeoning of methods that incorporate network analysis to improve understanding and prediction of complex phenotypes [16]. These approaches leverage information encoded in the interactions of proteins or genes, which are naturally modeled as graphs. Further, there has been an explosion of available data including large gene expression compendia [5, 19] and protein-protein interaction maps [34].

A core problem in this area has always been identifying groups of co-acting genes/proteins, which often manifest as a clique or dense subgraph in the resulting network. In this work, we consider the specific setting of identifying gene co-expression modules (or pathways) from large datasets, with a downstream objective of aiding the development of new therapies for human disease.

There is substantial evidence that drugs with genetic support are more likely to progress through the drug development pipeline [29]. Prior work has shown that approaches that consider genes' roles in biological networks can be robust to gene mapping noise [9], which might suggest alternative treatment avenues when a directly associated gene cannot be targeted.

Unfortunately, the membership of genes in modules and the relative strength of effect a module has on co-expression of its constituents are not directly observable. In gene co-expression analysis, what we are able to obtain is pairwise correlations for all genes in the organism [23]. Existing approaches rely on machine-learning to identify clusters in these data sets [9, 21]; here, we propose a new combinatorial model for the problem.

By modeling the observed gene expression data as a projection of a weighted bipartite graph representing gene-module membership and strength of expression for each module, we can represent the problem as a decomposition of the co-expression network into a collection of (potentially overlapping) weighted cliques (we call this Weighted Clique Decomposition).

While the resulting problem is naturally NP-hard, we demonstrate that techniques from parameterized algorithms enable efficient approaches when the number of modules is small. We present two parameterized algorithms for solving this problem[1]; both run in polynomial time in the network size, but have exponential dependence on the number of modules. As a first step towards practicality, we implement these methods[2] and provide preliminary experimental results on biologically-inspired synthetic networks with ground-truth modules derived from data on gene transcription factors and gene co-expression modules identified using

*This work was supported by the NIH R01 HG010067 and the Gordon & Betty Moore Foundation under awards GBMF4552 and GBMF4560.

[†]University of Utah, mcooley@cs.utah.edu
[‡]University of Colorado School of Medicine, greenescientist@gmail.com
[§]Hasso Plattner Institute, davis.issac@hpi.de
[¶]University of Pennsylvania, milton.pividori@pennmedicine.upenn.edu
[‖]University of Utah, sullivan@cs.utah.edu

[1]one of which restricts to integral edge weights
[2]code is available at https://github.com/TheoryInPractice/cricca

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

a machine-learning approach.

2 Motivating Biological Problem

Complex human traits and diseases are caused by an intricated molecular machinery that interacts with environmental factors. For example, although asthma has some common features such as wheeze and shortness of breath, research suggests that this highly heterogeneous disease is comprised of several conditions [37], such as childhood-onset asthma and adult-onset asthma, which present different prognosis and response to treatment, and also differ in their genetic risk factors [30]. Genome-wide association studies (GWAS) are designed to improve our understanding of how genetic variation leads to phenotype by detecting genetic variants correlated with disease. GWAS have prioritized causal molecular mechanisms that, when disturbed, confer disease susceptibility, and these findings were later translated to new treatments [35]. Drug targets backed by the support of genetic associations are more likely to succeed through the process of clinical development [29]. However, understanding the influence of genetic variation on disease pathophysiology towards the development of effective therapeutics is complex. GWAS often reveal variants with small effect sizes that do not account for much of the risk of a disease [31]. On the one hand, widespread gene pleiotropy (a gene affecting several unrelated phenotypes) and polygenic traits (a single trait affected by several genes) reveal the highly interconnected nature of biomolecular networks [25, 6].

Instead of looking at single gene-disease associations, methods that consider groups of genes that are functionally related (i.e., that belong to the same pathways) can be more robust to identify putative mechanisms that influence disease, and also provide alternative treatment avenues when directly associated genes are not druggable [22, 10]. Large gene expression compendia such as recount2 [5] or ARCHS4 [19] provide unified resources with publicly available RNA-seq data on tens of thousands of samples. Leveraging this massive amount of data, unsupervised network-based learning approaches [21, 32, 9] can detect meaningful gene co-expression patterns: sets of genes whose expression is consistently modulated across the same tissues or cell types. However, this is particularly challenging because the observed data is an aggregated and noisy projection of a highly complex transcriptional machinery: co-expressed genes can be controlled by the same regulatory program or module, but single genes can also play different roles in different modules expressed in distinct tissues or cell differentiation stages [2, 36]. For example, Marfan syndrome (MFS) is a rare genetic disorder caused by a mutation in gene *FBN1*, which encodes a protein that forms elastic and nonelastic connective tissue [28]. However, MFS is characterized by abnormalities in bones, joints, eyes, heart, and blood vessels, suggesting that *FBN1* is implicated in independent pathways across different tissues or cell types. In other words, the membership of genes in modules and the relative strength of effect a module has on the co-expression of its constituents are not directly observable from gene expression data.

3 Problem Modeling

We begin by observing that gene-module membership is naturally represented by a bipartite graph B, where each gene has an edge to all modules it participates in. Further, in order to capture the notion of varied effect-strength among modules, we associate a non-negative real-valued weight w_i to each module c_i, since we are interested in sets of co-expressed genes. In other words, we assume that all pairs of genes that are common to module i will be co-expressed with strength w_i; thus, the genes in each module will form a clique in the co-expression network. Further, we assume that modules interact with one another in a linear, additive manner. That is, the co-expression between genes u and v is the sum of the weights of all modules containing both u and v. In a noise-free setting, this means that the gene-gene co-expression network is exactly a union of (potentially overlapping) cliques $m_1, \dots m_k$ with associated weights $w_1, \dots w_k$ so that the weight on uv is exactly $\sum_{\{u,v\} \subseteq m_i} w_i$. It is important to note that not all valid solutions are interesting; specifically, one can always assign each pair of genes to its own clique of size 2, and get a valid solution. We rely on the principal of parsimony, and try to find an assignment which minimizes the number of modules in a valid solution. Realistically, the edge-weights will not satisfy exact equality, and we will need to consider an optimization variant of our problem which minimizes an objective function incorporating penalties for over/under-estimating the observed co-expressions.

To this end, we introduce a penalty function ϕ on the edges based on the discrepancy between the weight predicted by clique (module) membership and the original weight (observed co-expression value), then minimize ϕ to determine an optimal solution. For example, a natural choice for ϕ might be the sum of the absolute value of the discrepancies on each edge. Formally, this leads to the following problem:

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

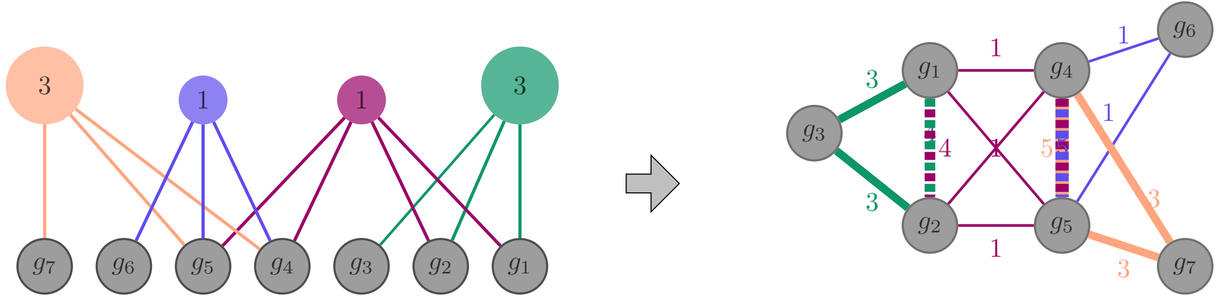

Figure 1: A bipartite graph (left) of genes $g_1, \ldots g_7$ and modules (top, labelled with strength of expression) gives rise to a gene-gene interaction network (right) with edges weighted by the sum of the strengths of all modules that contain both endpoints (indicated by color coding).

Weighted Clique Decomposition

Input: a graph $G = (V, E)$, a non-negative weight function w_e on E, a penalty function ϕ, and a positive integer k.

Output: a set of at most k cliques $C_1, \ldots C_k$ with weights $\gamma_1, \ldots \gamma_k \in \mathbb{R}^+$ that define $\gamma_{uv} = \sum_{i:uv \in C_i} \gamma_i$ for all $uv \in E$, such that $\phi(\{(w_e, \gamma_e) : e \in E\})$ is minimized.

In the remainder of this paper, we restrict our attention to the setting where equality can be achieved (as one might expect in synthetic data); further discussion of ideas for addressing the optimization variant is deferred to the future work section. For convenience, we define a decision version of WCD for this setting (this is equivalent to having a penalty function which is zero for matching the weight on an edge and infinite for any discrepancy):

Exact Weighted Clique Decomposition

Input: a graph $G = (V, E)$, a non-negative weight function w_e for $e \in E$, and a positive integer k.

Output: a set of at most k cliques $C_1, \ldots C_k$ with weights $\gamma_1, \ldots \gamma_k \in \mathbb{R}^+$ such that $w_{uv} = \sum_{i:uv \in C_i} \gamma_i$ for all $uv \in E$ (if one exists, otherwise output NO).

If the clique weights are constrained to be integers then the problem becomes a generalization of the NP-hard problem Edge Clique Partition [20, 13]. The NP-hardness of the fractional-clique-weight version also follows easily from the reduction in [20]. For completeness, we give the proof in Appendix F.

3.1 Annotated and Matrix Formulations We will work with the following more general version of EWCD in our algorithms, where some of the vertices are annotated with vertex weights.

Annotated EWCD

Input: a graph $G = (V, E)$, a non-negative weight function w_e for $e \in E$, a special set of vertices $S \subseteq V$, a non-negative weight function w_v for $v \in S$, and a positive integer k.

Output: a set of at most k cliques $C_1, \ldots C_k$ with weights $\gamma_1, \ldots \gamma_k \in \mathbb{R}^+$ such that $w_{uv} = \sum_{i:uv \in C_i} \gamma_i$ for all $uv \in E$ and $w_v = \sum_{i:v \in C_i} \gamma_i$ for all $v \in S$ (if one exists, otherwise output NO).

Note that EWCD is the special case of AEWCD when the set $S = \varnothing$.

We also introduce an equivalent matrix formulation of AEWCD, as our techniques are heavily based on linear algebraic properties. For this we use matrices that allow wildcard entries denoted by $\star$. For $a, b \in \mathbb{R} \cup \{\star\}$, we say $a \overset{\star}{=} b$ if either $a = b$ or $a = \star$ or $b = \star$. For matrices A and B, we say $A \overset{\star}{=} B$ if $A_{ij} \overset{\star}{=} B_{ij}$ for each i, j. We call the matrix problem as Binary Symmetric Weighted Decomposition with Diagonal Wildcards. Note that EWCD is the special case where all the diagonal entries are wildcards.

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

BSWD-DW

Input: a symmetric matrix $A \in \left(\mathbb{R}_0^+ \cup \{\star\}\right)^{n\times n}$ with wildcards appearing on a subset of diagonal entries, and a positive integer k

Output: a matrix $B \in \{0,1\}^{n\times k}$ and a diagonal matrix $W \in (\mathbb{R}_0^+)^{k\times k}$ such that $A \stackrel{\star}{=} BWB^T$. (if such (B,W) exist, otherwise output NO).

4 Parameterized Algorithms

Parameterized algorithms are a method used to tackle NP-hard problems where, besides the input size n, we are given an additional parameter k, most often representing the solution size. E.g., in our problem WCD, the parameter k is the number of cliques. An algorithm is said to be *fixed parameter tractable* if the runtime is polynomial in the input size and exponential only in the parameter—often resulting in tractable algorithms when the parameter is much smaller compared to the input size. One of the most effective tools in parameterized algorithms is *kernelization*, which is essentially a preprocessing framework that reduces the input to an equivalent instance of the same problem whose size depends only on the parameter k. The reduced instance is called a *kernel*. Sometimes, the reduction is not to the same problem itself but to a different related problem, in which case it is called a *compression*. For an extensive introduction to the topic, we refer to the book by Cygan et al. [8].

5 Prior Work

The WCD problem with integer clique weights is a generalization of the Weighted Edge Clique Partition problem which in turn generalizes Edge Clique Partition [20]):

Weighted Edge Clique Partition

Input: a graph $G = (V, E)$, a weight function $w_e : E \to Z^+$ and a positive integer k.

Output: a set of at most k cliques such that each edge appears in exactly as many cliques as its weight (if it exists, otherwise output NO).

Weighted Edge Clique Partition (WECP) was introduced by Feldmann et al. [12] last year. They gave a 4^k-compression and a $2^{\mathcal{O}(k^{3/2}w^{1/2}\log(k/w))} + \mathcal{O}(n^2 \log n)$ time algorithm for WECP, where w is the maximum edge weight. The compression is into a more general problem called Annotated Weighted Edge Clique Partition (AWECP) where some vertices also have input weights and these vertices are constrained to be in as many cliques as its weight in the output. The authors worked with an equivalent matrix formulation for AWECP called Binary Symmetric Decomposition with Diagonal Wildcards (BSD-DW) where given a $n \times n$ symmetric matrix A with wildcards (denoted by $\star$) in the diagonal, the task is to find a $n \times k$ binary matrix B such that $BB^T \stackrel{\star}{=} A$ where $\stackrel{\star}{=}$ denotes that the wildcards are considered equal to any number. The algorithm of Feldmann et al. [12] builds upon the linear algebraic techniques used by Chandran et al. [3] for solving the Biclique Partition problem. Our algorithms further build upon the techniques of [12]. Note that one could encode the clique weights (in the integer weight case) into the WECP problem by thinking of a clique of weight w as w identical unweighted cliques. This makes the parameter k equal to the sum of clique weights, and hence the algorithms of Feldmann et al. [12] are not sufficient for our application.

The unit-weighted case of WECP called Edge Clique Partition (ECP) has been more well studied, especially from the parameterized point of view. It is known that ECP admits a k^2-kernel in polynomial time [27]. The fastest FPT algorithm for ECP is the algorithm by Feldmann et al. [12] which runs in $2^{\mathcal{O}(k^{3/2}\log k)} + \mathcal{O}(n^2 \log n)$ for ECP. There are faster algorithms for ECP in special graph classes, for instance a $2^{\mathcal{O}(\sqrt{k})}n^{\mathcal{O}(1)}$ time algorithm for planar graphs, $2^{dk}n^{\mathcal{O}(1)}$ time algorithm for graphs with degeneracy d, and a $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time algorithm for K_4-free graphs [13]. A closely related problem to ECP is the Edge Clique Cover problem. Here, each edge should be present in at least one clique but can be present in any number of cliques. This unrestricted covering version is much harder and is known to *not* admit algorithms running faster than $2^{2^{o(k)}}n^{\mathcal{O}(1)}$ [7].

There are a few papers that study symmetric matrix factorization problems that are similar to the Binary Symmetric Decomposition with diagonal Wildcards (BSD-DW) problem, defined by Feldmann et al. [12]. Recall that BSD-DW is equivalent to the AWECP problem. Zhang et al. [38] studied the objective of minimizing $\|A - BB^T\|_2^2$. Their matrix model does not translate into the clique model as they do not have wildcards in the diagonal. Chen et al. [4] studied the objective of minimizing $\|A - BB^T\|_0$, but also without wildcards. A matrix model that has diagonal wildcards was considered by Moutier et al. [26] under the name Off-Diagonal Symmetric Non-negative Matrix Factorization, but they allow B to be any

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

non-negative matrix and not just binary.

The non-symmetric variants of these matrix problems known as Binary Matrix Factorization, have been receiving a lot of attention recently [24, 14, 15, 1, 18, 3]. Here the objective is to minimize $\|A - BC\|$, where A is an $m \times n$ input matrix, B is an $m \times k$ output binary matrix and C is a $k \times n$ output binary matrix. For example, a constant approximation algorithm running in $2^{\mathcal{O}(k^2 \log k)}(mn)^{\mathcal{O}(1)}$ is known [18]. In the graph-world the non-symmetric problems correspond to finding a partition of the edges of a bipartite graph into bicliques (complete bipartite graphs) instead of cliques [3].

6 Algorithms

We give two algorithms for BSWD-DW and hence also for the equivalent AEWCD and the special case EWCD. Both algorithms will have a common framework similar to that of Feldmann et al. [12]. The algorithms will differ in the method in which the clique weights (represented by the diagonal matrix W) are inferred. One uses an LP based method while the other uses an integer partition dynamic program.

The first step in our pipeline is to preprocess disjoint cliques and cliques that overlap only on single vertices (and thus have no overlapping edges) out of each graph. The specifics of this process are outlined in Appendix G, but it essentially runs a modified breadth-first search algorithm. Similar to Feldmann et al. [12], the second step in our algorithms is to give a kernel. The kernel follows the same reduction rules as in Feldmann et al. [12] i.e. by reducing blocks of twin vertices. The proof of correctness follows analogously, and we omit it here due to space constraints. After the kernelization, we can assume that the number of vertices of G (equivalently the number of rows of matrix A) is at most 4^k.

Theorem 6.1. *AEWCD* (BSWD-DW *resp.*) *has a kernel with at most* 4^k *vertices* (4^k *rows resp.*) *that can be found in* $\mathcal{O}(n^3)$ *time.*

The third step is to run a clique decomposition algorithm on the kernelized AEWCD instance to obtain the clique assignments for each vertex and clique weights. Let A be the input instance for BSWD-DW and let G be the corresponding input instance to AEWCD. Both our algorithms use the basis-guessing principle used by Feldmann et al. [12], first introduced by Chandran et al. [3]. The principle is that once we correctly guess the entries of a row-basis of B, then the remaining rows of B can be filled iteratively without backtracking. However, the technique does not carry over directly to the clique-weighted problem we have here. We additionally need to infer the clique-weight matrix W, which poses some additional challenges. Note that it is not feasible to guess the entries of the diagonals of W as each entry could be as large as the largest element in A. So once we have a guess for the basis, we also need to infer compatible values of W. Since there could be multiple choices for compatible W, we are not guaranteed to hit the correct solution for W, likely producing some backtracking while filling the non-basis rows. We tackle this by showing that if we guess a row-basis plus an additional k rows (thus at most $2k$ rows) then the choice of W does not matter. If our guess for this $2k$ rows (we call it the *pseudo-basis* of B) is correct, then we show that we can fill the other rows iteratively without any backtracking. The intuition of why we need the additional k rows is as follows: in the version without the W matrix, once we fix the basis $\widetilde{B}$ fo B, the matrix $\widetilde{A}$ given by the corresponding rows of A is fixed. In particular the diagonal values $\widetilde{A}_{ii}$ (that could have been wildcards and hence not fixed apriori) are now fixed. But with the matrix W, for different choices of W, we get different diagonal entries in $\widetilde{A}$. We only need to add at most one more row to the pseudo-basis in order to fix one of these diagonal entries. Thus we need at most k additional rows in the pseudo-basis.

We guess the rows of the pseudo-basis on-demand i.e., we add a row as basis-row only if a compatible row cannot be found for it under the current inferred clique weights W from the current basis matrix. Every time we add a new row to the basis, we recompute the clique weights W. The two algorithms that we present, differ in how they infer the clique weights for the current pseudo-basis. The first algorithm uses a linear programming method while the second uses an integer partitioning dynamic programming method. In our algorithms, we will often use partially filled matrices, i.e. some of the entries are allowed to have *null* values. If a row or matrix has all null values we call them null row and null matrix respectively.

6.1 Clique Weight Recovery by Linear Programming

We use a linear program to infer the clique weights for the current pseudo-basis of the algorithm. The pseudocode is given in `InferCliqWts-LP` (Algorithm 2). Suppose $\widetilde{B}$ is the current pseudo-basis matrix i.e. $\widetilde{B}$ is an $n \times k$ matrix where the current pseudo-basis rows (at most $2k$) are filled by 0's and $1'$s, and the other rows are null rows. For each pair of distinct non-null rows $\widetilde{B}_i$ and $\widetilde{B}_j$ we add the constraint $\widetilde{B}_i^T W \widetilde{B}_j = A_{ij}$ to the LP. Also, for each A_{ii} that is not a $\star$, we add the constraint $\widetilde{B}_i^T W \widetilde{B}_i = A_{ii}$. Note that the variables of the LP are the diagonal entries $W_{11}, W_{22}, \cdots, W_{kk}$. We also have non-negativity constraints $W_{11} \geq 0, \cdots, W_{kk} \geq 0$. Any feasible solution to this LP gives us a set of clique weights compatible with the current pseudo-basis. If

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

the LP is infeasible, then we conclude that the current pseudo-basis guess is infeasible and proceed to the next guess. Note that since we are only concerned about a feasible solution satisfying the constraints, we do not have an objective function for the LP. We point out that solving this LP is rather efficient as the number of variables are k and number of constraints are at most $4k^2$ and can be solved incrementally as we add constraints everytime a basis row is added.

ALGORITHM 1. `CliqueDecomp-LP`

1: **for** $P \in \{0,1\}^{2k \times k}$ **do**
2: initialize $\widetilde{B}$ to a $n \times k$ null matrix
3: $b, i \leftarrow 1$
4: **while** $b \leq 2k$ **do**
5: $\widetilde{B}_i \leftarrow P_b$
6: $b \leftarrow b + 1$
7: $W \leftarrow$ `InferCliqWts-LP` $(A, \widetilde{B})$
8: **if** W is not null matrix **then**
9: $(B, i) \leftarrow$ `FillNonBasis` (A, $\widetilde{B}$, W)
10: **if** $i = n + 1$ **then return** (B, W)
11: **else** $b \leftarrow 2k + 1$ ▷ *null W; break out of while*
12: **return** No

ALGORITHM 2. `InferCliqWts-LP` $(A, \widetilde{B})$

1: **let** $\gamma_1, \cdots, \gamma_k \geq 0$ be variables of the LP
2: **for** all pairs of non-null rows $\widetilde{B}_i, \widetilde{B}_j$ s.t. $A_{ij} \neq \star$ **do**
3: Add LP constraint $\sum_{1 \leq q \leq k} \widetilde{B}_{iq} \widetilde{B}_{jq} \gamma_q = A_{ij}$
4: **if** the LP is infeasible **then return** the null matrix
5: **else return** the diagonal matrix given by $\gamma_1, \ldots, \gamma_k$

ALGORITHM 3. `FillNonBasis` $(A, \widetilde{B}, W)$

1: $B \leftarrow \widetilde{B}$
2: **while** B has a null row **do**
3: **let** B_i be the first null row
4: **for** $v \in \{0,1\}^k$ **do**
5: **if** `iWCompatible` (A, B, W, i, v) **then**
6: $B_i \leftarrow v$
7: **goto** line 2
8: **return** (B, i) ▷ *there is no (i, W)-compatible v*
9: **return** $(B, n + 1)$ ▷ *B has no null row*

Once we have inferred a W compatible with the current pseudo-basis, we then try to fill the remaining rows (we call them non-basis rows) one by one in `FillNonBasis`. We say that a vector $v \in \{0,1\}^k$ is (i, W) compatible with row B_j if $v^T W B_j = A_{ij}$. We say that v is (i, W) compatible with matrix B if it

ALGORITHM 4. `iWCompatible` $(A, \widetilde{B}, W, i, v)$

1: **for** each non-null row B_j **do**
2: **if** $v^T W B_j \neq A_{ij}$ **then return** false
3: **if** $v^T W v \not\stackrel{\star}{=} A_{ii}$ **then return** false
4: **return** true

is (i, W)-compatible with each non-null row B_j, and $v^T W v \stackrel{\star}{=} A_{ii}$. We say B and W are compatible with each other if for each pair of non-null rows B_i and B_j, we have $B_i^T W B_j \stackrel{\star}{=} A_{ij}$. We keep filling the rows B_i of B one-by-one with (i, W)-compatible rows until either B is completely filled or there is an i such that there is no (i, W)-compatible vector in $\{0,1\}^k$. In the former case we show that (B, W) gives a solution, and in the latter case we proceed on to take row i into the pseudo-basis row. Note that when we take a new row into the pseudo-basis row we throw away all the non-basis rows and make them null rows again. We will show that we only need to take up to $2k$ rows into the pseudo-basis for the algorithm to correctly find a solution.

6.1.1 Algorithm Correctness

THEOREM 6.2. *CliqueDecomp-LP (Algorithm 1) correctly solves the BSWD-DW problem, and hence also correctly solves AEWCD and EWCD, in time $\mathcal{O}(4^{k^2} k^2 (32^k + k^3 L))$, where L is the number of bits required for input representation.*

First we prove in the following lemma that if `CliqueDecomp-LP` outputs Yes, i.e. if it outputs through line 10, then the matrices B and W output indeed satisfy that $A \stackrel{\star}{=} BWB^T$. The proof follows because we checked for (i, W)-compatibility whenever we filled B_i. The full proof of the Lemma can be found in Appendix B.1.

LEMMA 6.1. *If CliqueDecomp-LP returns through line 10, then the matrices B and W output satisfy that $A \stackrel{\star}{=} BWB^T$.*

Lemma 6.1 immediately implies that if the instance is a No-instance then the algorithm does not output through line 10. Since the only other possibility for output is through Line 12, which outputs No, we can conclude that for a No-instance we correctly output No. The following arguments are therefore related to the correctness of Yes instances.

For arguing the correctness in the Yes case, we fix a valid solution (B^*, W^*) of the instance. If the output occurs through Line 10, then by Lemma 6.1, we are done. So for the sake of contradiction assume that the output does not occur through Line 10. For $I \subseteq [n]$, we

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

define B_I^* as the $n \times k$ matrix whose i-th row is equal to B_i^* for all $i \in I$ and the other rows are null rows. The following lemma follows because we iterate over all possible values of pattern matrix P.

Lemma 6.2. *Let $I \subseteq [n]$ be such that $|I| \leq 2k-1$. If $\widetilde{B}$ is equal to B_I^* at some point in the algorithm, and if* **FillNonBasis** *$(A, \widetilde{B} = B_I^*, W)$ called in Line 9 returns $i \leq n$, then $\widetilde{B}$ is equal to $B_{I\cup\{i\}}^*$ at some point in the algorithm.*

So, if we start with $I = \varnothing$, and repeatedly apply Lemma 6.2, then at some point in the algorithm, we have $\widetilde{B} = B_I^*$ such that $|I| = 2k$. Towards this, we define the matrix $E(\widetilde{B})$ formed by the rows that are element-wise products of pairs of non-null rows in $\widetilde{B}$. More precisely:

Definition 6.1. *$E(\widetilde{B})$ is the matrix containing rows $\widetilde{B}_i \odot \widetilde{B}_j$ for each pair i, j (not necessarily distinct) such that $A_{ij} \neq \star$. Here $\odot$ denotes element-wise product.*

The above definition means that $E(\widetilde{B})$ is the coefficient matrix of the LP that the algorithm would construct in the call `InferCliqWts-LP` $(A, \widetilde{B})$.

Definition 6.2. (Pseudo-rank) *The pseudo-rank of $\widetilde{B}$ is defined as the sum of ranks of $\widetilde{B}$ and $E(\widetilde{B})$, where by rank of $\widetilde{B}$ we mean the rank of the matrix formed by the non-null rows of $\widetilde{B}$.*

Since the number of columns in $\widetilde{B}$ and $E(\widetilde{B})$ are each k, we have the following lemma.

Lemma 6.3. *The pseudo-rank of $\widetilde{B}$ is at most $2k$.*

We say that a vector $v \in \{0,1\}^k$ *i-extends* $\widetilde{B}$ if $\widetilde{B}_i$ is currently a null row, and adding v as $\widetilde{B}_i$ increases the pseudo-rank of $\widetilde{B}$.

Lemma 6.4. *If* **FillNonBasis** *$(A, \widetilde{B} = B_I^*, W)$ called on Line 9 returns $i \leq n$, then B_i^* i-extends B_I^*.*

Proof. Suppose for the sake of contradiction that B_i^* does not i-extend B_I^*. This means that B_i^* is linearly dependent on the non-null rows of B_I^* and each $B_i^* \odot B_j^*$ for $j \in I$ is linearly dependent on the rows of $E(\widetilde{B})$. Also, if $A_{ii} \neq \star$ then $B_i^* \odot B_i^*$ is linearly dependent on the rows of $E(\widetilde{B})$. Now, consider each non-null row B_j of the matrix B when `iWCompatible` (A, B, W, i, v) was called in Line 5 in `FillNonBasis`. We prove that $B_i^{*T} W B_j = A_{ij}$ and that $B_i^{*T} W B_i^* \stackrel{\star}{=} A_{ii}$. This then implies that B_i^* is (i, W)-compatible with B and hence `FillNonBasis` could not have returned i, giving a contradiction.

First consider the case when B_j is a pseudo-basis row, i.e. $j \in I$. Since B_i^* does not i-extend B_I^*, we know $B_i^* \odot B_j^*$ is linearly dependent on the rows of $E(\widetilde{B})$. This means that adding B_i^* as $\widetilde{B}_i$ would not add any linearly independent equality constraints to the LP system that solves for W. So, either the LP becomes infeasible or all the solutions to the LP still remain solutions. But the LP is not infeasible as the diagonal elements of W^* gives a feasible solution to the LP. Thus, the current W remains a feasible solution even after the addition of B_i^* the row $\widetilde{B}_i$. Hence, $B_i^{*T} W B_j^* = A_{ij}$.

Now, consider the case when B_j is not a pseudo-basis row, i.e. $j \notin I$. In other words $\widetilde{B}_j$ is a null row and B_j was added in `FillNonBasis`. Since B_i^* does not i-extend B_I^*, we know that B_i^* is linearly dependent on the non-null rows of B_I^*. In other words, $B_i^* = \sum_{\ell \in I} \lambda_\ell B_\ell^*$ where each $\lambda_\ell \in \mathbb{R}$. Then,

$$B_i^{*T} W B_j = \sum_{\ell \in I} \lambda_\ell B_\ell^{*T} W B_j \tag{6.1}$$

$$= \sum_{\ell \in I} \lambda_\ell A_{\ell j} \tag{6.2}$$

$$= A_{ij} \tag{6.3}$$

where Eq. (6.2) is because B_j could have been selected for row j only if it was (j, W)-compatible with B_I^*, and Eq. (6.3) follows by using that $W^* B^*$ is a linear map from B^* to A and hence the linear dependencies in B^* are preserved in A. More precisely,

$$\begin{aligned} \sum_{\ell \in I} \lambda_\ell A_{\ell j} &= \sum_{\ell \in I} \lambda_\ell B_\ell^{*T} W^* B_j^* \\ &= B_i^* W^* B_j^* \\ &= A_{ij} \end{aligned}$$

Note that we have here crucially used $\ell \neq j$ (as $j \notin I$) and $j \neq i$ to say $=$ and not just $\stackrel{\star}{=}$. This is the reason we required a separate argument for $j \in I$. The argument for $B_i^{*T} W B_i^* \stackrel{\star}{=} A_{ii}$ follows the same argument as in the case of $j \in I$ by observing that if $A_{ii} \neq \star$ then $B_i^* \odot B_i^*$ is linearly dependent on the rows of $E(\widetilde{B})$. ∎

Now starting with $I = \varnothing$, and applying Lemmas 6.2 and 6.4 repeatedly, we have that at some point in the algorithm, $\widetilde{B}$ is equal to some B_I^* such that the pseudo-rank of B_I^* is $2k$. At this point, the `FillNonBasis` call at Line 9 should return $i = n+1$ because if it returned $i \leq n$, then adding B_i^* to $\widetilde{B}$ would make the pseudo-rank of $\widetilde{B}$ equal to $2k+1$, a contradiction to Lemma 6.3. Hence, the algorithm outputs through Line 10. This concludes the correctness of the algorithm. We defer the runtime analysis to Appendix C.1.

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

6.2 Clique Weight Recovery by Integer Partitioning

We give an algorithm for inferring the current pseudo-basis's clique weights by solving an integer partitioning dynamic program. The pseudocode is given in `InferCliqWts-IP` (Algorithm 6). Similar to 6.1, consider $\widetilde{B}$, the current pseudo-basis matrix. Additionally, a list $\mathbf{W}$ is maintained, containing partially filled diagonal weight matrices each of which are *compatible* with the current pseudo-basis matrix $\widetilde{B}$. Here compatibility is defined as follows. For a matrix B, first define its *relevant indices*, denoted by $R(B)$, as the set of all $r \in [k]$ such that there exist non-null rows B_i, B_j such that $B_{ir}B_{jr} = 1$ and $A_{ij} \neq \star$. For a diagonal matrix W we define $F(W)$ as the set of all $i \in [k]$ such that W_{ii} is not null. We say that $\widetilde{B}$ and W are compatible if $F(W) = R(\widetilde{B})$ and for each pair of non-null rows $\widetilde{B}_i, \widetilde{B}_j$ such that $A_{ij} \neq \star$, it is true that $\sum_{r \in R(\widetilde{B})} \widetilde{B}_{ir} W_{rr} \widetilde{B}_{jr} = A_{ij}$. We maintain in $\mathbf{W}$, all the possible fillings of indices $R(B)$ of the diagonal vector of clique-weight matrix W such that W is compatible with $\widetilde{B}$.

`InferCliqWts-IP` is given as input the current $\widetilde{B}$ after initially inserting P_b at Line 6 in `CliqueDecomp-IP`. Thus, $\widetilde{B}_i$ is the potential basis row we are considering. At the start of the function call, we know that each non-null rows $\widetilde{B}_j$ is (j, W)-compatible with each non-null row $\widetilde{B}_{j'}$ for $j, j' \neq i$. If $\widetilde{B}_i$ is not (i, W)-compatible, this implies that either there are null weights in W in relevant positions, or the current basis P being considered is not the correct one. Let $\widetilde{B}_j$ be a non-null row such that $\widetilde{B}_i$ is not (i, W)-compatible with $\widetilde{B}_j$. We define $X \subseteq [k]$ as the indices of null positions in the diagonal of W, $\overline{\mathbf{X}} = [k] \setminus X$, and $P \subseteq [k]$ as the set of positions in $\widetilde{B}_i \odot \widetilde{B}_j$ having 1 values. The sum $t = \sum_{l \in P \cap \overline{\mathbf{X}}}^{k} \widetilde{B}_{il} \widetilde{B}_{jl} W_{ll}$ is the sum of all previously fixed clique-weights that contribute to A_{ij}. The difference $s = t - A_{ij}$ has to be contributed by $P \cap X$. The `UpdateWs` function finds all possible ways to sum to s using $|P \cap X|$ number of non-negative integers via a dynamic program. This is an integer partitioning problem and is a simple variant of the common change-making problem. Then for each such combination a new W-matrix is created by inserting the combination in the indices $P \cap X$. `UpdateWs` returns the list of all such W-matrices created. Note that s could be negative in which case `UpdateWs` returns an empty list.

6.2.1 Algorithm Correctness

THEOREM 6.3. *`CliqueDecomp-IP` (Algorithm 5) correctly solves the BSWD-DW problem, and hence also correctly solves AEWCD and EWCD, in time $\mathcal{O}(4^{k^2} 32^k w^k k)$ where w is the maximum entry of A.*

ALGORITHM 5. `CliqueDecomp-IP`

1: **for** $P \in \{0,1\}^{2k \times k}$ **do**
2: $\quad \widetilde{B} \leftarrow n \times k$ null matrix
3: $\quad b, i \leftarrow 1$
4: $\quad \mathbf{W} \leftarrow \{$null matrix$\}$
5: $\quad$ **while** $b \leq 2k$ **do**
6: $\quad\quad \widetilde{B}_i \leftarrow P_b$
7: $\quad\quad b \leftarrow b + 1$
8: $\quad\quad \mathbf{S} \leftarrow$ `InferCliqWts-IP` $(A, \widetilde{B}, \mathbf{W}, i)$
9: $\quad\quad$ **if** $\mathbf{S}$ is not empty **then**
10: $\quad\quad\quad \mathbf{W} \leftarrow \mathbf{S}$
11: $\quad\quad\quad (B, i) \leftarrow$ `FillNonBasis` $(A, \widetilde{B}, W[0])$
12: $\quad\quad\quad$ **if** $i = n + 1$ **then return** $(B, W[0])$
13: $\quad\quad$ **else**
14: $\quad\quad\quad b \leftarrow 2k + 1$
15: **return** No

ALGORITHM 6. `InferCliqWts-IP` $(A, \widetilde{B}, \mathbf{W}, i)$

1: $\mathbf{S} \leftarrow []$
2: **for** $l \leftarrow 1$ to $|\mathbf{W}|$ **do**
3: $\quad \mathbf{X} \leftarrow \{x \in [k] \mid W[l]_{xx}$ is null$\}$
4: $\quad \overline{\mathbf{X}} \leftarrow [k] \setminus X$
5: $\quad$ **temp** $\leftarrow$ empty queue
6: $\quad$ **temp**.push($\mathbf{W}[l]$)
7: $\quad$ **for** each non-null row $\widetilde{B}_j$ s.t. $A_{ij} \neq \star$ **do**
8: $\quad\quad$ *iters* $\leftarrow |$**temp**$|$
9: $\quad\quad \mathbf{P} \leftarrow \{p \in [k] \mid \widetilde{B}_{ip}\widetilde{B}_{jp} = 1\}$
10: $\quad\quad$ **for** $q \leftarrow 1$ to *iters* **do**
11: $\quad\quad\quad S' \leftarrow$ **temp**.pop()
12: $\quad\quad\quad s \leftarrow A_{ij} - \sum_{f \in \mathbf{P} \cap \overline{\mathbf{X}}} S'_{ff}$
13: $\quad\quad\quad \mathbf{T} \leftarrow$ `UpdateWs` $(S', \mathbf{P} \cap \mathbf{X}, s)$
14: $\quad\quad\quad$ **temp**.push($\mathbf{T}$)
15: $\quad\quad\quad$ **if temp** is empty **then** goto Line 16
16: $\quad \mathbf{S}$.push(**temp**)
17: **return S**

ALGORITHM 7. `UpdateWs` $(W, \mathbf{I}, s)$

1: $\mathbf{V} \leftarrow [\,]$
2: $\mathbf{Y} \leftarrow$ all partitions of s into $|I|$ non-negative integers
3: **for** each *parts* in $\mathbf{Y}$ **do**
4: $\quad y \leftarrow 0$
5: $\quad C \leftarrow W$
6: $\quad$ **for** each $i \in \mathbf{I}$ **do**
7: $\quad\quad C_{ii} \leftarrow parts[y]$
8: $\quad\quad y \leftarrow y + 1$
9: $\quad \mathbf{V}$.push(C)
10: **return V**

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

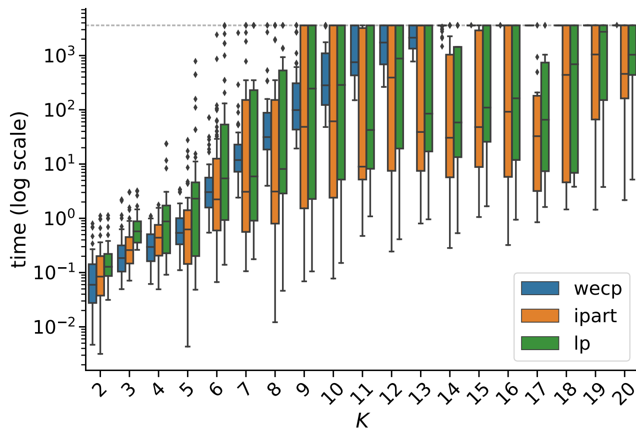

Figure 2: Log-scale plot showing distribution of total algorithm runtimes when binned by K (the sum of the clique weights). All K values shown in Figure 8 in Appendix E.

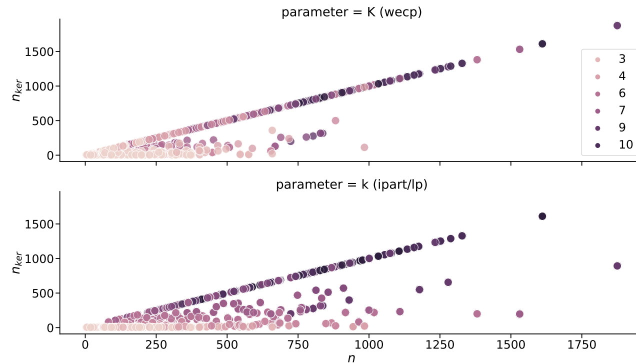

Figure 3: Instance size reduction due to kernelization (from n to n_{ker}); points along the diagonal experienced no reduction from kernel rules. Top shows reduction using parameter K, bottom shows reduction when using parameter k.

The proof of `CliqueDecomp-IP`'s correctness is similar to the proof of `CliqueDecomp-LP` in Section 6.1.1. We describe the differences in Appendix B.2 and defer the runtime analysis to Appendix C.2.

7 Experimental Setup

This section describes the synthetic corpora; hardware descriptions can be found in Appendix E.2. We generate two sets of biologically-inspired synthetic graphs. The first dataset defines modules (cliques) using known relationships between transcription factors and genes; the second uses latent variables from a machine learning approach for analyzing co-expression data.

7.1 TF-Dataset Our first dataset emulates the bipartite gene-module network by using known relationships between transcription factors (TFs) and genes [11]. To generate a network with a ground truth of k cliques, we randomly select k TFs and form the network which is the union of all associated genes with edges between those that share at least one selected TF.

Since the relative strengths of effect on expression are unknown, we specify a desired maximum edge weight (see Appendix D.1), and generate integral clique weights as described in Appendix D.2. A heavy-tailed distribution is chosen to mimic the view that modules have widely varying effects on gene co-expression, and a small minority likely have drastically higher impact than all others [32].

7.2 LV-Dataset A similar approach is taken when generating the set of the latent variable-associated synthetic graphs. In this data [17, 32], each latent variable (LV) represents a set of genes that are co-expressed in the same cell types. A score for every gene in each LV indicates the strength of its association to the module. Further, some latent variables have been shown to align with prior knowledge of pathway associations [32]. Our generator randomly selects k latent variables, with 80% drawn from those known to be aligned with pathways, and the remaining 20% chosen uniformly from all LVs. For each LV, we only include genes with association scores above a threshold, determined as described in Appendix D.3.

In contrast to the TF data, here the clique weights have a basis in the underlying data. For each LV, we compute the average associate score over all included genes then linearly transform this to control the maximum edge weight in the network (see Appendix D.1).

8 Results

This section highlights the key outcomes of our preliminary experimental evaluation. We begin by highlighting the effects of reparameterization, comparing the algorithm of [12] (referred to as `wecp`) to `CliqueDecomp-IP` and `CliqueDecomp-LP` (shortened to `ipart` and `lp` for consistency with figure legends).

8.1 Effects of Reparameterization To compare the effects on the runtime of reparameterizing from the sum of the clique weights K to the number of distinct cliques k, we tested `wecp`, `ipart`, and `lp` on all instances with $k \leq 11$ and small/medium weight scalings. As seen in Figure 2, and Figures 5, 6, 8 in Appendix E, both algorithms parameterized by k are faster across the entire corpus when $k >= 6$. It should be noted that the slower `ipart` and `lp` runtimes for $k < 6$ are partly

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

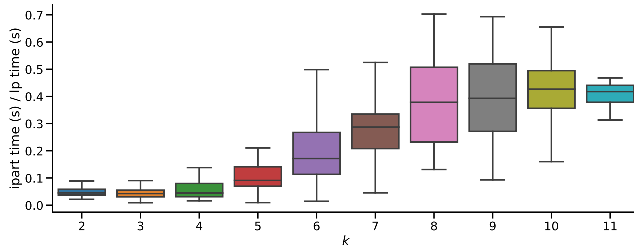

Figure 4: Relative runtimes of `ipart` and `lp` on full corpus with $2 \leq k \leq 11$ and all weight scalings; times exclude kernel (which is shared). Outliers not shown.

due to the preprocessing often removing highly weighted cliques, effectively setting $k = K$. When $k = K$, it is slightly faster to run `wecp` due to not having to compute the clique weights. Figure 2 shows the runtimes for $K \in [2, 20]$, and Figure 8 shows all $K \in [2, 49]$. After $K = 13$ `wecp` timedout for every instance, whereas `ipart` and `lp` are still able to compute solutions in under an hour.

Some performance increase may be attributed to the kernelization, as shown in Figure 3. Since one of the reduction rules relies on the parameter, the instance size after kernelization (denoted n_{ker}) is different between `wecp` (which uses K) shown in the top figure, and `ipart`/`lp` (which use k) shown in the bottom figure. Figure 3 shows that the kernel gives great reduction when parameterized by k when k is small and has less of an impact when k is large. Figure 6 in Appendix E shows a comparison of the runtimes when instances are sorted by the size of the kernelized instance.

8.2 Integer Partitioning vs LP From Figure 2, we observe that `ipart` runs slightly faster on average than `lp`, which was unexpected. To further compare the two, we ran both algorithms on a larger corpus including all instances with $k \leq 11$ regardless of weight scaling. Figure 4 shows the runtime ratio of the two algorithms[3]. We observe that despite a consistent advantage for `ipart` on small k, the methods' runtimes seem to converge as we approach $k = 9$ and we hypothesize that `lp` will become the dominant approach for larger k when testing with a longer timeout than one hour. To test whether the specific clique-weight assignment mattered, we evaluated runtimes on 6 random weight assignment permutations for each instance. Table 4 in Appendix E.3 shows that both algorithms are virtually unaffected by the assignments.

Finally, since `ipart`'s complexity depends on the maximum weight w, we evaluated this effect by comparing performance across the small, medium, and large weight-scale variants of each instance. Figure 10 in Appendix E shows that the relative increase in runtime between weight scalings is fairly consistent across algorithms, indicating minimal effect. However, it is noteworthy that `ipart` experiences much higher variance.

[3]these times exclude the shared kernel to emphasize the difference in the two approaches

8.3 Ground Truth While these algorithms are guaranteed to find a decomposition using at most k weighted cliques if one exists, a unique solution is not guaranteed. For the synthetic corpus, we verified that our output recovered the LVs/TFs selected by the generators, but we do not know whether this will generalize to real data (where weight distributions may be quite different) or much larger k. Additionally, Appendix E.4 analyzes the impact on incorrect k input on the recovered solution.

9 Conclusions & Future Work

This paper offers a new combinatorial framing of the problem of module identification in gene co-expression data, Weighted Clique Decomposition. Further, we present two new parameterized algorithms for the noise-free setting (EWCD), removing the dependence of a prior approach on the magnitude of the clique weights. To address concerns of practicality, we implement both approaches and evaluate them on a corpus of biologically-inspired genetic association data. The empirical results show that both new approaches significantly outperform the Weighted Edge Clique Partition algorithm of [12], and that worst-case asymptotic runtime bounds are not realized on typical inputs.

While these algorithms provide a nice first step towards a polynomial-time algorithm for module-identification, there remain many unaddressed challenges before use on real data. While our exponential dependence on the number of modules is to be expected from an FPT algorithm, in many real-world datasets there are hundreds of underlying functional groups. One way to overcome this limitation is to incorporate a hierarchical approach, which would provide an extra biologically meaningful outcome: the relationships between cliques could be mapped to representations such as the Gene Ontology, where descendents represent more specialized function.

We would like to extend our approach to the non-exact setting, targeting approximation schemes for the optimization variant of the problem. Understanding the correct penalty function for under- and over-estimating edges, and whether this should be amplified for edges below the threshold in the original coexpression data will be critical in informing a useful technique.

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

A Kernel for AEWCD

Here, we give the 4^k-kernelization for BSWD-DW (and hence also for the equivalent AEWCD), and prove its correctness and runtime. Let (A, k) be the input BSWD-DW instance and let (G, w_e, w_v, k) be the corresponding AEWCD instance. For a vertex u of G we will use A_u and B_u to denote their corresponding rows in matrices A and B respectively. Similarly we use A_{uv} for an element of the matrix A corresponding to the pair of vertices u and v.

First, we divide the vertices of input graph G (correspondingly the rows of input matrix A) into *blocks*, as follows. We say that i and j are in the same block if $A_i \stackrel{\star}{=} A_j$. Note that $\stackrel{\star}{=}$ is an equivalence relation over the rows of A, as proven by Feldmann et. al. [12, Lemma 7]. Then, we apply the following two reduction rules exhaustively.

Rule 1. *If there are more than 2^k blocks then output that the instance is a NO-instance.*

Rule 2. *If there is a block D of size greater than 2^k, then pick two distinct $i, j \in D$. We reduce to an instance (A', k) as follows: $G' := G - (D \setminus \{i\})$, $A'_{ii} = A_{ij}$, and $A'_{uv} = A_{uv}$ for all $(u, v) \in (V(G') \times V(G')) \setminus \{(i, i)\}$. Given a solution (B', W') for (A', k) we construct a solution for (A, k) as $W = W'$, $B_u = B'_u$ for all $u \notin D$, and $B_u = B'_i$ for all $u \in D$.*

Once the two rules are applied exhaustively, then the reduced instance has size at most 4^k because there are at most 2^k blocks by Rule 1 and each block size is at most 2^k by Rule 2. So, it only remains to prove that the two reduction rules are correct, and also to prove the runtime of the kernelization. The following lemma gives the correctness of Rule 1.

Lemma A.1. *If (A, k) is a YES-instance of BSWD-DW, then there are at most 2^k blocks in A.*

Proof. Suppose there are more than 2^k blocks. Let (B, W) be a solution. Since there are only 2^k distinct binary vectors, there exist i and j in different blocks such that $B_i = B_j$. Then we have $A_i \stackrel{\star}{=} B_i^T W B_i = B_j^T W B_j \stackrel{\star}{=} A_j$. This implies $A_i \stackrel{\star}{=} A_j$ (because if $x \stackrel{\star}{=} y \stackrel{\star}{=} z$ and y does not contain any $\star$ then $x \stackrel{\star}{=} z$), and hence i, j and j are in the same block, a contradiction. ∎

Now, we prove the correctness of Rule 2 in the following two lemmas.

Lemma A.2. *In Rule 2, if the reduced instance (A', k) has a solution (B', W') then the solution (B, W) constructed by Rule 2 is indeed a solution to (A, k).*

Proof. It is sufficient to prove that $B_u^T W B_v \stackrel{\star}{=} A_{uv}$ for all $u, v \in V(G)$. First consider the case when $u, v \notin D$, the block picked by Rule 2. Then $B_u^T W B_v = {B'_u}^T W B'_v \stackrel{\star}{=} A'_{uv} = A_{uv}$. Now, consider the case when $u \in D, v \notin D$. Then $B_u^T W B_v = {B'_u}^T W B'_i \stackrel{\star}{=} A'_{ui} = A_{ui}$. Finally, consider the case when $u \in D, v \in D$. We can assume $A_{uv} \neq \star$ as this case follows trivially. Then $B_u^T W B_v = {B'_i}^T W B'_i \stackrel{\star}{=} A'_{ii} = A_{ij} = A_{uv}$. Here, the last equality is because any two entries (that are not $\star$) in the same block of matrix A are equal [12, Lemma 7]. ∎

Lemma A.3. *In Rule 2, if (A, k) is a YES-instance then the reduced instance (A', k) is a YES-instance.*

Proof. Let (B, W) be a solution of (A, k). Since the block D contains more than 2^k rows, there exist row indices p and q such that $B_p = B_q$. We define a solution (B', W) for (A', k) as $B'_u := B_u$ for all $u \in V(G') \setminus \{i\}$ and $B'_i := B_p$.

To prove that (B', W) is indeed a valid solution for (A', k), it is sufficient to show that ${B'_u}^T W B'_v \stackrel{\star}{=} A'_{uv}$ for all $u, v \in V(G')$. First consider the case when $u, v \neq i$. Then ${B'_u}^T W B'_v = {B_u}^T W B_v \stackrel{\star}{=} A_{uv} = A'_{uv}$. Now, consider the case when $u = i, v \neq i$. Then ${B'_i}^T W B_v = {B_p}^T W B_v \stackrel{\star}{=} A_{pv} = A_{iv} = A'_{iv}$, where the second-to-last equality followed as p and i are in the same block D. Finally, consider the case when $u = v = i$. Then ${B'_i}^T W B'_i = {B_p}^T W B_p = B_p^T W B_q = A_{pq} \stackrel{\star}{=} A_{ij} = A'_{ii}$. Here, the $\stackrel{\star}{=}$ follows because any two entries (that are not $\star$) in the same block of matrix A are equal [12, Lemma 7]. Also, note that the third equality is an equality (and not only a '$\stackrel{\star}{=}$ equivalence') as A_{pq} is not a diagonal entry. ∎

It is rather easy to see that the runtime of the kernelization is $\mathcal{O}(n^3)$. The division into blocks can be easily realized in $\mathcal{O}(n^3)$ time. The application of Rule 1 then takes only $\mathcal{O}(1)$ time. Rule 2 is applied at most once to each block and all the applications together take only $\mathcal{O}(n^2)$ time.

B Algorithm Correctness

B.1 `CliqueDecomp-LP` Here we give the missing proofs for the correctness of algorithm `CliqueDecomp-LP`.

Proof. [Proof of Lemma 6.1] Each row of B is either a pseudo-basis row that was filled in Line 5 of `CliqueDecomp-LP` or it is a non-basis row that was filled in Line 6 of `FillNonBasis`. Now consider two pseudo-basis rows B_i and B_j. For them, we have $B_i^T W B_j = A_{ij}$ as W was a solution to the LP that contained the constraint $B_i^T W B_j = A_{ij}$. Also, for a pseudo-bais row B_i,

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

we have that $B_i^T B_i \stackrel{\star}{=} A_{ii}$ because if $A_{ii} \neq \star$, then we added the constraint $B_i^T W B_i = A_{ii}$ to the LP. Now consider a non-basis row B_i and some other row B_j that was filled before B_i. Note that B_j could be a pseudo-basis row or a non-basis row. Since the algorithm filled B_i in Line 6 of `FillNonBasis`, we know that B_i is (i, W)-compatible with all the rows filled before. Thus $B_i^T W B_j = A_{ij}$. Moreover, by (i, W)-compatibility, we also have $B_i^T W B_i \stackrel{\star}{=} A_{ii}$. Hence, we have that $BWB^T \stackrel{\star}{=} A$. ∎

B.2 `CliqueDecomp-IP` As noted in the main text, the majority of the proof of `CliqueDecomp-IP`'s correctness follows the proof of `CliqueDecomp-LP` in Section 6.1.1, so here we only point out the differences.

Since the main difference with the LP algorithm is the `InferCliqWts-IP` routine, we prove the correctness of it in the following lemma. The lemma follows directly from the construction of `InferCliqWts-IP` and `UpdateWs` algorithms.

LEMMA B.1. *Consider the function call* `InferCliqWts-IP`*$(A, \widetilde{B}, \boldsymbol{W}, i)$ in Line 8 of* `CliqueDecomp-IP`*. Let $\widetilde{B}'$ be the matrix $\widetilde{B}$ before the insertion of current row $\widetilde{B}_i$, i.e. $\widetilde{B}'_i$ is a null row. Suppose $\boldsymbol{W}$ contains all the compatible W with $\widetilde{B}'$ then the list $\boldsymbol{S}$ returned contains all the compatible W with $\widetilde{B}$.*

Once we have the above lemma, the correctness of the algorithm follows more or less the same proof as that of the LP algorithm. We state the following two key lemmas that are counterparts of Lemma 6.1 and Lemma 6.4.

LEMMA B.2. *If* `CliqueDecomp-IP` *returns through line 12, then the matrices B and $W[0]$ output satisfy that $A \stackrel{\star}{=} BW[0]B^T$.*

LEMMA B.3. *If* `FillNonBasis` *$(A, \widetilde{B} = B_I^*, W[0])$ called on Line 11 of* `CliqueDecomp-IP` *returns $i \leq n$, then B_i^* i-extends B_I^*.*

Both the lemmas follow the same proof as that of their counterparts. To derive Lemma B.3 from the proof of Lemma 6.4, it is sufficient to observe that the statement, *all solutions of the LP still remain solutions*, can be interpreted as all compatible W's still remain compatible. This is the reason why we need only to check with $W[0]$ in `FillNonBasis` and not with all matrices in the list $\mathbf{W}$. In fact, this is what we do even in the LP; the LP has many possible solutions but we check with only one solution. The difference is that the many solutions are implicitly captured by the LP constraint system, whereas `CliqueDecomp-IP` explicitly maintains all compatible combinations.

C Runtime Analysis

Here, we estimate the runtimes of `CliqueDecomp-LP` and `CliqueDecomp-IP`. Note that the runtimes we give are after kernelization, i.e. the input to these two algorithms are assumed to be a kernel according to Theorem 6.1. The kernelization incurs an additional runtime additive factor of $\mathcal{O}(n^3)$.

C.1 `CliqueDecomp-LP`

LEMMA C.1. `CliqueDecomp-LP` *(Algorithm 1) runs in time $\mathcal{O}(4^{k^2}k^2(32^k + k^3 L))$, where L is the number of bits required for input representation.*

Proof. The `for` loop in Line 1 has at most 2^{2k^2} iterations. The `while` loop in Line 4 has at most $2k$ iterations. The only steps that take more than unit time in the `while` loop are the calls to `InferCliqWts-LP` and `FillNonBasis`. `InferCliqWts-LP` solves an LP with k variables and at most $4k^2$ constraints.This can be solved in at least $\mathcal{O}(k^4 L)$ time by using standard algorithms [33]. It only remains to estimate the runtime of `FillNonBasis`. The `while` loop in Line 2 of `FillNonBasis` has at most n iterations and the `for` loop in Line 4 has at most 2^k iterations. The only non-trivial step in the `for` loop is the call to `iWCompatible`. The `for` loop in Line 1 of `iWCompatible` has at most n iterations and Line 2 takes at most k operations. Thus the time taken for `iWCompatible` is at most $\mathcal{O}(nk)$ and the time taken for `FillNonBasis` is at most $\mathcal{O}(n^2 k 2^k)$. Hence, the time taken for `CliqueDecomp-LP` is in $\mathcal{O}(2^{2k^2} 2k(k^4 L + n^2 k 2^k))$. The claimed run-time in the theorem follows by putting $n \leq 4^k$ due to the kernel. ∎

C.2 `CliqueDecomp-IP`

LEMMA C.2. `CliqueDecomp-IP` *(Algorithm 5) runs in $\mathcal{O}(4^{k^2} 32^k w^k k)$, where w is the maximum weight value in A.*

Proof. The `for` loop in Line 1 has at most 2^{2k^2} iterations. Let y be the number of distinct partially filled weight matrices. We have $y \leq (w+2)^k$ since each such matrix is defined by the k entires along its diagonal, each of which is either null or an integer from 0 to w.

Now, we show that for a fixed iteration of `for` loop in Line 1 of `CliqueDecomp-IP`,

1. each line of `CliqueDecomp-IP` in the `for` loop is executed at most y times,
2. each line in `InferCliqWts-IP` is executed at most y times (across all calls to the function from the fixed iteration of for loop), and

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

3. the `for` loop in Line 3 of `UpdateWs` is executed at most y times (across all calls to `UpdateWs` from all calls of `InferCliqWts-IP` from the fixed iteration of for loop in `CliqueDecomp-IP`).

For (1) observe that the list **W** in `CliqueDecomp-IP` always gets new matrices whenever it is modified. At any point the elements of the list are disjoint from the past entries of the list (during the fixed iteration of for loop). For (2) observe that between the two consecutive executions of a line, **temp** would have seen a new matrix that was not in it before. For (3) observe that every time a new matrix is pushed to **V**, it is a new matrix that it did not have before.

From Appendix C.1, we know that `FillNonBasis` runs in $\mathcal{O}(n^2 k 2^k)$ time. All the other non-loop lines in `CliqueDecomp-IP`, `InferCliqWts-IP` and `UpdateWs` can be done in at most $\mathcal{O}(k^2)$ time. The for loop in Line 6 takes only $\mathcal{O}(k)$ time.

Therefore, the total running time of `InferCliqWts-IP` is

$$\begin{aligned}&\mathcal{O}(2^{2k^2} \cdot w^k \cdot (n^2 k 2^k + k^2))\\ &= \mathcal{O}(4^{k^2} 2^k w^k n^2 k)\\ &= \mathcal{O}(4^{k^2} 32^k w^k k)\end{aligned}$$

where the last equality follows by using that $n \leq 4^k$ after kernelization. □

	LV			TF		
k	#	n	m	#	n	m
2	60	129.3	6325.2	64	36.4	1221.7
3	60	229.3	17571.3	67	67.7	2315.7
4	60	352.6	32644.5	65	83.0	2750.1
5	60	387.8	41867.6	65	111.8	5434.5
6	60	445.0	42498.8	69	175.0	16745.6
7	60	605.5	92604.6	65	113.0	3212.9
8	60	661.6	89747.0	63	222.9	15585.9
9	60	713.7	110952.1	72	233.8	19834.6
10	60	737.2	74706.4	67	269.9	17520.0
11	60	823.1	77260.8	63	222.7	13646.4
12	45	876.8	112389.7	70	272.3	20116.5
13	42	872.9	69691.5	63	361.8	35362.8
14	45	859.9	69564.9	55	286.7	10791.9
15	27	954.6	66220.1	59	386.4	38712.8
16	12	848.2	45401.5	65	319.9	18656.3
17	-	-	-	43	361.0	16164.7
18	6	897.5	44828.5	37	354.6	24494.3
19	21	1042.7	57171.1	24	345.0	13114.5
20	3	1029.0	41588.0	40	340.4	13205.0

Table 1: Average instance sizes (number of nodes n and edges m) across k values for TF and LV datasets. The number of graphs for a given k value may be smaller than 60 if some instances were eliminated for not having $2-11$ cliques after preprocessing.

D Synthetic Data Specifications

Here we detail the parameter settings and methodology for the synthetic corpus generation. Each instance is generated by specifying a random seed, a desired number of cliques k, one of three weight scaling factors, and an underlying dataset (TF or LV). For each combination of parameters selected, we used 20 random seeds. We used all k values in $[2, 20]$, resulting in an initial corpus of 2280 graphs. We only ran experiments on those instances which had k values in $[2, 11]$ after pre-processing (see Appendix G), resulting in a final corpus of 1917 networks. Table 1 summarizes the average number of nodes and edges for the generated corpus.

Table 2 summarizes statistics about how the ground truth cliques overlapped across the entire corpus. For example, about six percent of the graphs (119 of the 1917) contained at least one clique which overlapped almost all ($81-100\%$) of the remaining cliques, and over ten percent (201) have their average clique overlapping over 40% of all the other cliques. Alternatively, if you measure entanglement of cliques (modules) by the percentage of their nodes (genes) that are shared with at least one other clique, we can see that fifteen percent (286) have some clique which shares more than 80 percent of its nodes with another clique, and just less than four percent (70 graphs) have an average overlap greater than 40% for all their cliques.

D.1 Clique Weight Scaling: In both corpora, we use three different scale factors (which we refer to as small, medium, and large) to control the maximum edge weight in the resulting networks. In the TF data, this takes the form of three different maximum edge weight values for our heavy-tailed weight generator: 1, 4, and 16. In the LV data, we scale the average gene-LV association scores by 1, 2, and 4. When this results in a non-integral weight; we take the ceiling.

D.2 TF Weight Generation As noted in Section 7.1, the transcription factor dataset provides no inherent strength of association for each TF. Given the belief that real data follows a heavy-tailed distribution, we generate random integer weights that mimic this (to the extent possible, given the extremely small number of cliques to be assigned weights) as follows.

We take as input a desired maximum weight Δ, and define three intervals $L = [1, \ell\Delta]$, $M = [m_l\Delta, m_r\Delta]$,

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

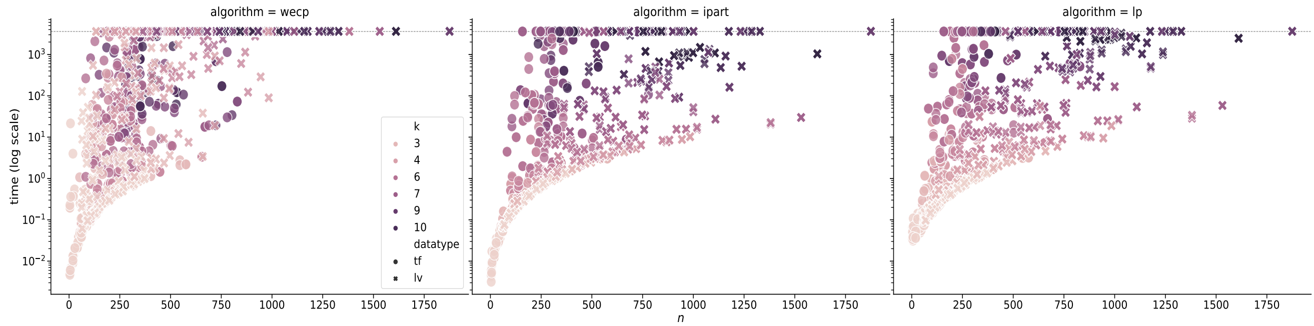

Figure 5: Log-scale runtimes (kernel + decomposition) of `wecp`, `ipart`, and `lp` on corpus of all TF (circle) and LV (star) instances with $2 \leq k \leq 11$ and small/medium weight scalings, sorted by instance size (prior to running the kernel).

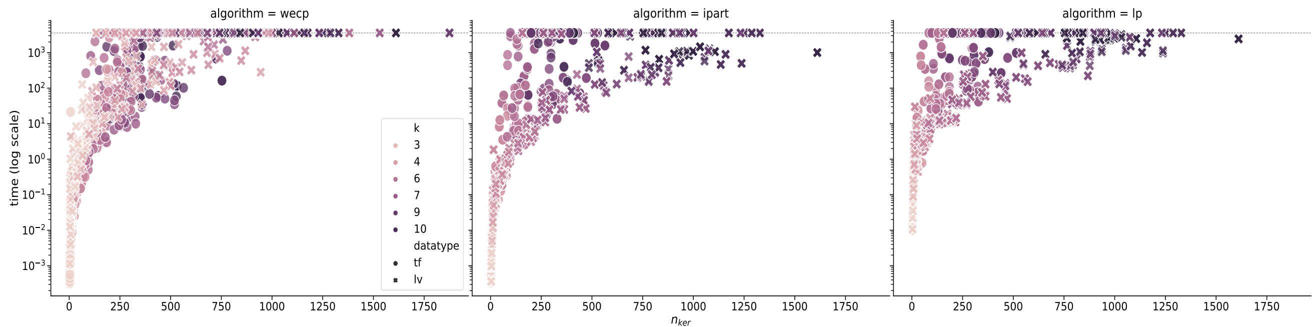

Figure 6: Log-scale runtimes (decomposition only) of `wecp`, `ipart`, and `lp` on corpus of all TF (circle) and LV (star) instances with $2 \leq k \leq 11$ and small/medium weight scalings, sorted by reduced instance size (after running the kernel).

$H = [h\Delta, \Delta]$. We set $\ell = .14$, $m_1 = .20$, $m_2 = .28$, and $h = .90$, ensuring the ranges are well-separated. To create a heavy-tail, we assign different probabilities to a weight being drawn from each range, $p_L = .75$, $p_M = .15$, and $p_R = .1$. Once an interval is selected, the weight is chosen uniformly at random among integers in its range. Each clique weight is generated from this process independently at random, with the caveat that we ensure that some clique receives weight Δ (to avoid instances with no high-valued clique).

D.3 LV Membership Thresholding The LV data includes an association score for each gene-LV pair; in order to identify strongly-associated genes to be included in the clique generated from a given LV, we use a uniform threshold. In order to determine an appropriate threshold for maintaining some variability of clique-size without including large numbers of spurious associations, we computed an elbow plot showing the number of genes per latent variable. This resulted in a threshold of 0.6, which was used for all instances.

E Supplemental Experimental Results

In this section, we provide additional experimental results. Note that in all data and analysis (in the appendix and main paper), k is referring to the parameter value of the instances *after* preprocessing.

E.1 Additional Reparametrization Effects Figure 5 shows the combined runtime of the kernel and decomposition algorithms on each instance sorted by the instance size before kernelization and colored by k value. `ipart` (middle) and `lp` (right) roughly solve instances with the same k value (regardless of n) in a similar amount of time. This is shown by the light to dark gradient from the bottom to the top of the figures. Whereas `wecp` (left) has no discernible gradient, meaning there is no connection between k and the running time of this algorithm. This is as expected since the input parameter to `wecp` is K and not k.

Figure 6 is similar to Figure 5, but instead shows the running time of only the decomposition algorithms sorted by the instance size after kernelization and colored by k value. When compared to Figure 5, these

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

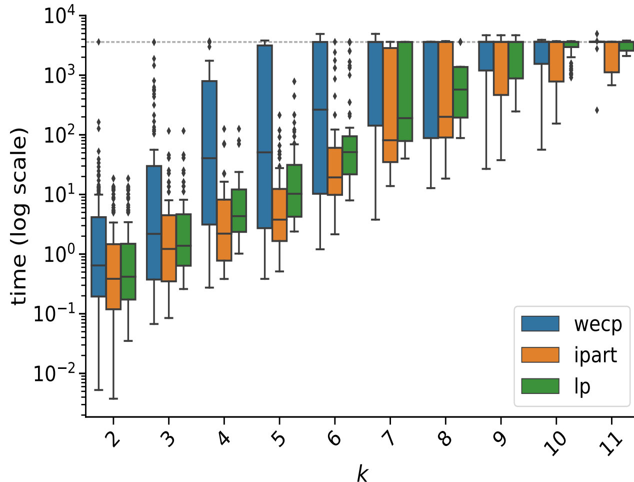

Figure 7: Log-scale distribution of total runtimes (preprocessing + kernel + decomposition) binned by k on all TF and LV instances with $2 \leq k \leq 11$ for small/medium weight scalings.

figures show that the kernel algorithm dramatically reduces the size of instances with small k and has a negligible effect on instances with large k values. The light to dark gradient is again apparent in the `ipart` (middle) and `lp` (right) figures, meaning the decomposition runtimes are dependent mainly on k. There is a slight increase in running time as n_{ker} increases as well. While the running time increases as n_{ker} increases for `wecp` (left), again, the colors are sporadic. `wecp` timed out on 337 instances, whereas `lp` timed out on 204 instances and `ipart` timed out on 171 instances (out of 986 total instances) indicated by the points running across the top lines in both figures.

Figure 7 shows the total runtime of preprocessing, kernelization, and decomposition binned by k. This figure shows that the median runtime of `wecp` across all k values is larger than both `ipart` and `lp` and has more variability. The median runtime of `ipart` is also less than `lp`. Figure 8 shows the runtime of the kernel and decomposition algorithms binned over all K values. `ipart` and `lp` can compute solutions up to $K = 44$, whereas `wecp` consistently times out once $K \geq 13$.

Table 3 shows the average peak memory usage for graphs with medium and large weight scales for the entire corpus (combining both TF and LV graphs). We observe that the `ipart` algorithm's average memory usage is slightly larger than the `lp` algorithm's, but the difference is not substantial. This contrasts with the `ipart` algorithm's large theoretical bound on the number of stored weight matrices. All k values for both algorithms have a standard deviation around 25.

	Node Overlap			Clique Overlap		
	min.	avg.	max.	min.	avg.	max.
0-20%	1911	1279	722	1785	917	311
21-40%	6	568	397	108	787	405
41-60%	0	70	266	21	201	672
61-80%	0	0	246	3	12	410
81-100%	0	0	286	0	0	119

Table 2: Summary of overlap between cliques across the corpus. At left, we report on the number of nodes shared between cliques in each graph (as a percent of n). The first column (min) reports the number of graphs where every clique had at least the given amount of overlap, the second (avg) reports based on the average, and the third (max) gives the number of graphs where the clique with the most overlap fell into the given range. At right, we use the same min, avg, max criterion, but instead measure overlap by the percentage of other cliques sharing at least one vertex.

k	`ipart`	`lp`
2	126.97	124.00
3	122.15	119.15
4	129.10	126.09
5	132.28	129.23
6	134.00	131.01

Table 3: Average peak memory usage in megabytes between the `ipart` and `lp` algorithms for k between $[2-6]$. Data are combined for medium and large weight scales for both TF and LV datasets.

Peak memory usage was tracked for each run of the algorithms using the python `resource` module.

E.2 Hardware All experiments used identical hardware; each machine runs Arch Linux version 3.10.0 − 957.27.2.el7.x86_64, have 40 Intel(R) Xeon(R) Gold 6230 CPUs (2.10GHz), and have 192GB of memory. All code is written in Python 3.

E.3 Varying Clique Weights To test if clique weight assignments affect the runtime of the `ipart` and `lp` algorithms, we randomly permuted the assignment of the same set of clique weights to the cliques of each TF graph six times. We ran both the `ipart` and `lp` algorithms for each weight assignment and recorded the runtimes. To compare runtimes for each k value, we first normalized all runtimes using min-max normalization. Using these normalized runtimes, we computed the difference between the maximum for a particular

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

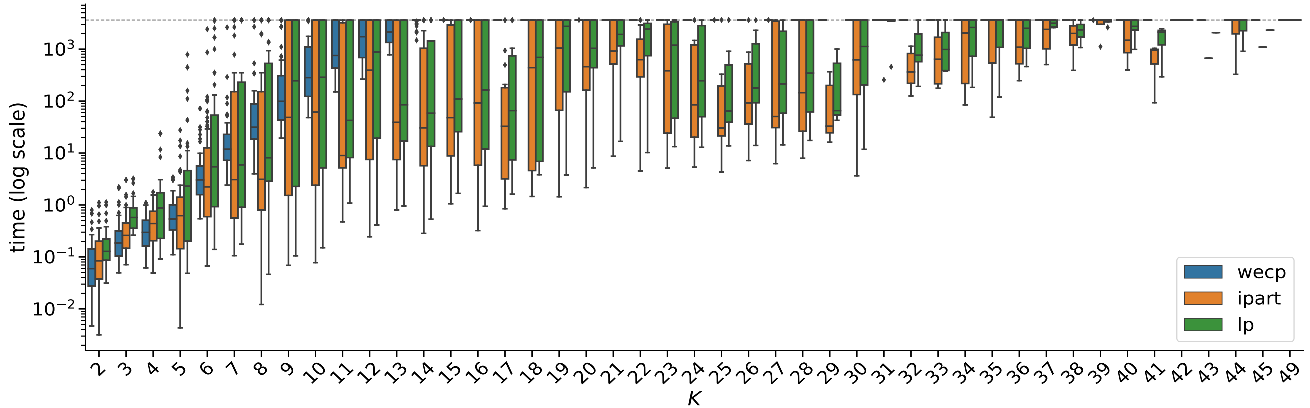

Figure 8: Log-scale plot showing distribution of total algorithm runtimes (kernel + decomposition) when binned by K for all K values.

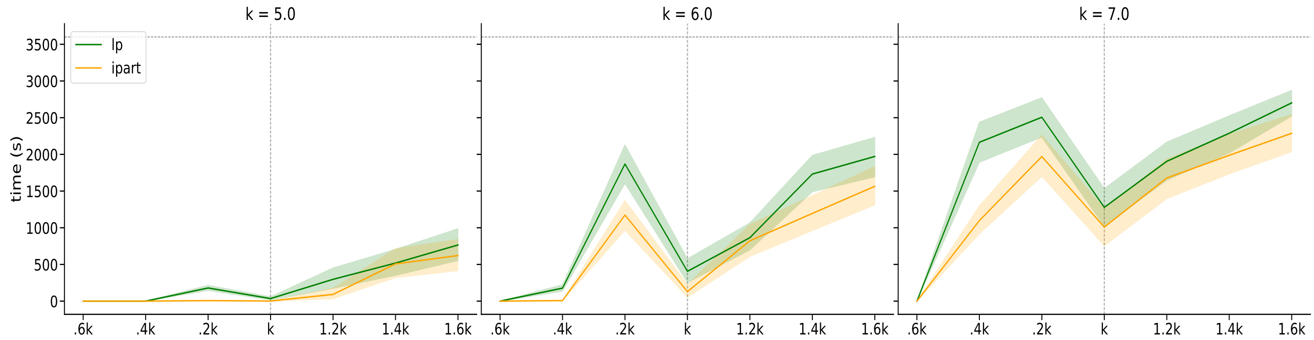

Figure 9: Average runtimes (kernel + decomposition) between all instances with ground truth $k = 5, 6, 7$ when inputting k values in the range $[0.6k, 0.4k, 0.2k, k, 1.2k, 1.4k, 1.6k]$. Vertical dashed line indicates average time for ground-truth k as input.

k	ipart	lp
5	0.001 (0.004)	0.000 (0.001)
6	0.019 (0.060)	0.016 (0.066)
7	0.004 (0.012)	0.071 (0.189)
8	0.000 (0.000)	0.275 (0.287)

Table 4: Average difference (and standard deviation) between the maximum and minimum normalized runtimes when clique weight assignments were permuted six times per TF graph. Data are grouped by k. The runtime differences when $k \in [2, 4]$ were all too small to be measured.

weight assignment and the minimum for each graph, then averaged across all graphs with the same k value. Table 4 shows these results. Since the differences are quite small, we conclude that specific clique weight assignments have little effect on the runtime of both the `ipart` and `lp` algorithms.

E.4 Performance When k is Unknown In practice, the ground truth number of distinct cliques (modules) in real-world graphs is unknown. We tested the effect of incorrect input of this parameter value on the runtime and solution quality. Figure 9 shows the runtime ratio (including the kernel time) across the same instances when inputing k not equal to the ground truth value. Inputting incorrect k values for both `ipart` and `lp` results in an large increase in runtime. When $k = 0.6k$ (i.e., a much smaller value than the true k), the kernel outputs a No answer, thus the runtime of the decomposition algorithms is always zero. Interestingly, for all instances with $k \in [0.6k, 0.4k, 0.2k]$, no solution was recovered, and for $k \in [k, 1.2k, 1.4k, 1.6k]$, all ground truth solutions were recovered, even when the input k value is larger than the ground truth.

F NP-hardness of EWCD

We use the same construction as given by [20]. Using this, we will show that EWCD is NP-hard even when

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

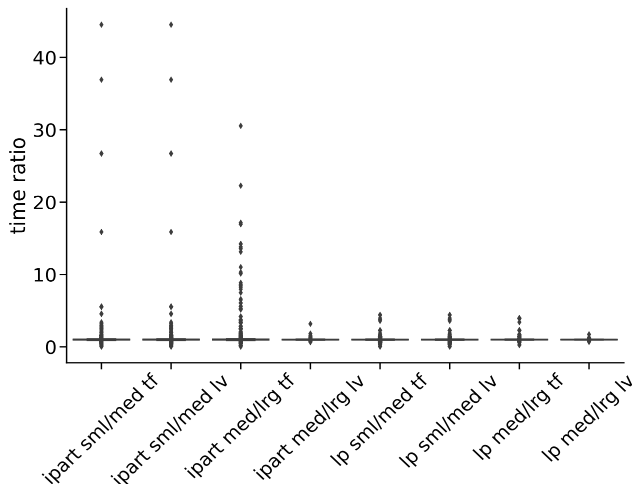

Figure 10: Runtime ratio of `ipart` and `lp` when run on the same underlying instance with increased weight scaling factor: small vs medium and medium vs large.

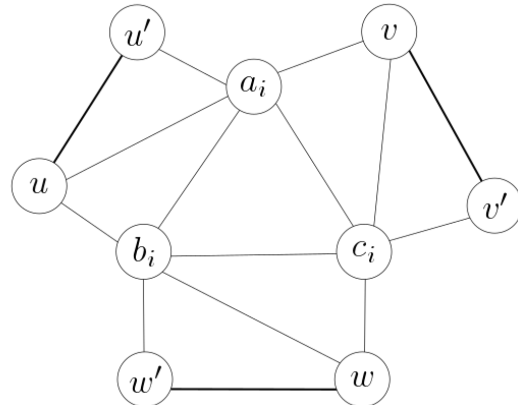

Figure 11: The gadget for set $S_i = \{u, v, w\}$

restricted to K_4-free graphs and unit edge-weights. They use a reduction from the NP-hard Exact 3-Cover (E3C). In this problem, we are given a universe U of $3q$ elements and a collection $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of m 3-ary subsets of U. The decision problem is whether there exist q sets that cover all the elements. Note that if there is such a covering then each element is covered exactly once in the solution.

Given an instance of E3C, we construct an instance of EWCD on a K_4-free graph G as follows. For each element $u \in U$, we will have an edge uu' in G. We call these edges as the *element-edges*. For each set $S_i = \{u, v, w\}$ in $\mathcal{S}$, we will have three vertices a_i, b_i, c_i that form a triangle. We connect this triangle to the edges uu', vv' and ww' as shown in Fig. 11. All edges have weight 1. We set the budget k for EWCD to be $6m + q$.

First we show that if the E3C instance has a solution then so does the EWCD instance. Without loss of generality assume that $S_1, S_2, \ldots S_q$ is a solution to E3C. Then for each $1 \le i \le q$, we take the 7 cliques $\{u, u', a_i\}$, $\{v, v', c_i\}$, $\{w, w', b_i\}$, $\{a_i, b_i, c_i\}$, $\{u, b_i\}$, $\{v, a_i\}$, and $\{w, c_i\}$ into the solution. For each $q + 1 \le i \le m$, we take the 6 cliques $\{u, a_i, b_i\}$, $\{v, a_i, c_i\}$, $\{w, c_i, b_i\}$, $\{u', a_i\}$, $\{v', c_i\}$, and $\{w', b_i\}$ into the solution. We give each clique a weight of 1. It is easy to check that this gives a valid solution for EWCD using exactly $k = 6m + q$ cliques.

Now, we show that if the EWCD instance has a solution then so does the E3C instance. Let $\mathcal{C}$ be the set of cliques in the solution to EWCD. Note that the cliques' weights could be fractional. However, each edge has to be present in at least one of the cliques in $\mathcal{C}$. Let E_i denote the set of edges in the gadget corresponding to set S_i that are exclusively in the gadget (the 12 thin edges in Fig. 11), that is, for $S_i = \{u, v, w\}$, the set $E_i = \{a_ib_i, a_iu, ub_i, a_ic_i, a_iv, c_iv, b_ic_i, b_iw, wc_i, a_iu', c_iv', b_iw'\}$. Let $\mathcal{C}_i$ be the set of all cliques in $\mathcal{C}$ which contain at least one edge from E_i. It is easy to see that at least 6 distinct cliques are required to cover the edge set E_i. Hence, $|\mathcal{C}_i| \ge 6$. Also, it is easy to see that $\mathcal{C}_i \cap \mathcal{C}_j = \varnothing$ for distinct i, j.

Lemma F.1. *If a clique $C \in \mathcal{C}_i$ contains an element-edge then $|\mathcal{C}_i| \ge 7$.*

Proof. Let $S_i = \{u, v, w\}$. Without loss of generality, let the element-edge contained in C be ww'. Then $C = \{w, w', b_i\}$. Then $\mathcal{C}_i$ contains the K_2 $\{c_i, w\}$. This is because the only other clique that could cover the edge c_iw is $\{b_i, c_i, w\}$; but this clique can have a weight strictly less than 1 as otherwise the total weight of the cliques containing edge b_iw exceeds 1. Further, the edges in $E_i \setminus \{b_iw', b_iw, wc_i\}$ require at least 5 cliques to cover, proving $|\mathcal{C}_i| \ge 7$. □

Since k is only $6m + q$, the above lemma implies that there are q indices $i \in [m]$ such that $\mathcal{C}_i$ covers 3 element-edges. Taking the sets S_i for these q indices gives the required solution for E3C.

G Preprocessing Specification

`CliqueDecomp-LP` and `CliqueDecomp-IP` both have runtime exponential in the number of cliques (k). Pruning away easily detectible cliques before running the expensive decomposition algorithms reduces the size of the input parameter, thus reducing the overall runtime.

The preprocessing works by running a modified breadth-first search algorithm to detect and remove cliques that are either (1) disjoint from the rest of the

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

k	n	m	k	cliques removed
2	92.74%	92.74%	92.74%	1.85
3	67.45%	65.09%	72.97%	2.19
4	65.78%	62.98%	73.60%	2.94
5	51.08%	46.98%	66.56%	3.33
6	36.53%	31.41%	53.75%	3.22
7	44.59%	36.08%	62.29%	4.36
8	36.30%	30.74%	52.95%	4.24
9	36.43%	31.47%	52.95%	4.77
10	19.08%	10.78%	43.23%	4.32
11	26.95%	17.94%	47.67%	5.24
12	21.33%	11.32%	47.76%	5.73
13	20.89%	12.98%	47.40%	6.16
14	16.32%	6.27%	46.29%	6.48
15	14.99%	6.14%	45.53%	6.83
16	25.05%	15.15%	54.22%	8.67
17	20.74%	9.61%	56.95%	9.68
18	16.78%	6.44%	52.27%	9.41
19	18.41%	7.51%	49.47%	9.40
20	24.64%	14.60%	59.33%	11.87

Table 5: Percentage reduction in instance size (n, m) and number of cliques (k) after preprocessing. TF and LV data are combined and grouped by initial k value (prior to preprocessing). Average (absolute) number of cliques removed is also reported.

network (i.e. in their own connected component) or (2) intersect other cliques exclusively on single vertices (i.e., share no edges with other cliques). In any valid decomposition, a clique C of size ℓ with edges of weight w from either of these categories must be represented by between one and $\ell(\ell-1)/2$ cliques, all with weight w. Thus, if there is any solution with k cliques, there must also be a solution which includes C (with weight w) and has at most k cliques. Thus, removing C from the instance and reducing k by 1 will not change whether or not we have a yes-instance (and C can be added to the resulting set to form a valid solution for the original instance). Because our algorithm depends exponentially on k, this pre-processing results in much faster overall runtimes.

Our algorithm first loops over each $v_i \in V$ checking if all its incident edges have the same weight. If they do, call this edge weight w_i and U_i the set of vertices adjacent to v_i. We then iterate over each $u_j \in U_i$ and check that there exists an edge from u_j to u_k for each $u_k \in U_i$ with edge weight equal to w_i. If this process returns true, then the set $U_i \cup v_i$ forms a clique. Since all edge weights are equal, the clique does not share any edges with other cliques, making it safe for removal. This process takes $O(VE)$ time. In our experiments, preprocessing took an average of 4.67 seconds to run on the TF graphs, and 33.71 seconds on the LV graphs. Table 5 shows the average reduction in instance sizes and clique counts.

References

[1] F. Ban, V. Bhattiprolu, K. Bringmann, P. Kolev, E. Lee, and D. P. Woodruff. A PTAS for ℓ_p-low rank approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 747–766. Society for Industrial and Applied Mathematics, 2019.

[2] William S. Bush, Matthew T. Oetjens, and Dana C. Crawford. Unravelling the human genome-phenome relationship using phenome-wide association studies. *Nature Reviews Genetics*, 17(3):129–145, 2016. URL: https://doi.org/10.1038/nrg.2015.36.

[3] S. Chandran, D. Issac, and A. Karrenbauer. On the parameterized complexity of biclique cover and partition. In *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[4] S. Chen, Z. Song, R. Tao, and R. Zhang. Symmetric boolean factor analysis with applications to instahide. arxiv. preprint, 2021. arXiv:2102.01570.

[5] Leonardo Collado-Torres, Abhinav Nellore, Kai Kammers, Shannon E Ellis, Margaret A Taub, Kasper D Hansen, Andrew E Jaffe, Ben Langmead, and Jeffrey T Leek. Reproducible RNA-seq analysis using recount2. *Nat. Biotechnol.*, 35(4):319–321, 2017.

[6] Heather J. Cordell. Detecting gene-gene interactions that underlie human diseases. *Nature Reviews Genetics*, 10(6):392–404, 2009. URL: https://doi.org/10.1038/nrg2579.

[7] M. Cygan, M. Pilipczuk, and M. Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM Journal on Computing*, 45(1):67–83, 2016.

[8] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

[9] Christiaan A. de Leeuw, Joris M. Mooij, Tom Heskes, and Danielle Posthuma. Magma: Generalized gene-set analysis of gwas data. *PLOS Computational Biology*, 11(4):e1004219, 2015. doi:10.1371/journal.pcbi.1004219.

[10] Mikhail G. Dozmorov, Kellen G. Cresswell, Silviu-Alin Bacanu, Carl Craver, Mark Reimers, and Kenneth S. Kendler. A method for estimating coherence of molecular mechanisms in major human disease and traits. *BMC Bioinformatics*, 21(1):473, 2020. doi:10.1186/s12859-020-03821-x.

[11] Ahmed Essaghir, Federica Toffalini, Laurent Knoops, Anders Kallin, Jacques van Helden, and Jean-Baptiste Demoulin. Transcription factor regulation

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

can be accurately predicted from the presence of target gene signatures in microarray gene expression data. *Nucleic Acids Research*, 38(11):e120–e120, 03 2010. arXiv:https://academic.oup.com/nar/article-pdf/38/11/e120/16764928/gkq149.pdf, doi:10.1093/nar/gkq149.

[12] A. E. Feldmann, D. Isaac, and A. Rai. Fixed-parameter tractability of the weighted edge clique partition problem. In *15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[13] R. Fleischer and X. Wu. Edge clique partition of k 4-free and planar graphs. In *International Conference on Computational Geometry, Graphs and Applications . , , Heidelberg*, pages 84–95, Berlin, 2010. Springer.

[14] F. V. Fomin, P. A. Golovach, D. Lokshtanov, F. Panolan, and S. Saurabh. Approximation schemes for low-rank binary matrix approximation problems. *ACM Transactions on Algorithms (TALG)*, 16(1):1–39, 2019.

[15] F. V. Fomin, P. A. Golovach, and F. Panolan. Parameterized low-rank binary matrix approximation. *Data Mining and Knowledge Discovery*, 34(2):478–532, 2020.

[16] Casey S. Greene, Arjun Krishnan, Aaron K. Wong, Emanuela Ricciotti, Rene A. Zelaya, Daniel S. Himmelstein, Ran Zhang, Boris M. Hartmann, Elena Zaslavsky, Stuart C. Sealfon, Daniel I. Chasman, Garret A. FitzGerald, Kara Dolinski, Tilo Grosser, and Olga G. Troyanskaya. Understanding multicellular function and disease with human tissue-specific networks. *Nature Genetics*, 47(6):569–576, 2015. URL: https://doi.org/10.1038/ng.3259.

[17] Qiwen Hu and Jaclyn Taroni. Multiplier - latent variables extracted from recount2. https://figshare.com/articles/dataset/recount_rpkm_RData/5716033/4, 2018.

[18] R. Kumar, R. Panigrahy, A. Rahimi, and D. Woodruff. Faster algorithms for binary matrix factorization. In *International Conference on Machine Learning*, pages 3551–3559. PMLR, 2019.

[19] Alexander Lachmann, Denis Torre, Alexandra B. Keenan, Kathleen M. Jagodnik, Hoyjin J. Lee, Lily Wang, Moshe C. Silverstein, and Avi Ma'ayan. Massive mining of publicly available rna-seq data from human and mouse. *Nature Communications*, 9(1):1366, 2018. URL: https://doi.org/10.1038/s41467-018-03751-6.

[20] SH Ma, WD Wallis, and JL Wu. The complexity of the clique partition number problem. *Congr. Numer*, 67:59–66, 1988.

[21] Weiguang Mao, Elena Zaslavsky, Boris M. Hartmann, Stuart C. Sealfon, and Maria Chikina. Pathway-level information extractor (plier) for gene expression data. *Nature Methods*, 16(7):607–610, 2019. URL: https://doi.org/10.1038/s41592-019-0456-1.

[22] Jörg Menche, Amitabh Sharma, Maksim Kitsak, Susan Dina Ghiassian, Marc Vidal, Joseph Loscalzo, and Albert-László Barabási. Uncovering disease-disease relationships through the incomplete interactome. *Science*, 347(6224):1257601, February 2015. URL: http://science.sciencemag.org/content/347/6224/1257601.abstract, doi:10.1126/science.1257601.

[23] Daniele Mercatelli, Laura Scalambra, Luca Triboli, Forest Ray, and Federico M. Giorgi. Gene regulatory network inference resources: A practical overview. *Biochimica et Biophysica Acta (BBA) - Gene Regulatory Mechanisms*, 1863(6):194430, 2020. URL: https://www.sciencedirect.com/science/article/pii/S1874939919300410, doi:10.1016/j.bbagrm.2019.194430.

[24] P. Miettinen and S. Neumann. Recent developments in boolean matrix factorization. preprint, 2020. arXiv:2012.03127.

[25] Jason H. Moore, Folkert W. Asselbergs, and Scott M. Williams. Bioinformatics challenges for genome-wide association studies. *Bioinformatics*, 26(4):445–455, 2010. URL: https://doi.org/10.1093/bioinformatics/btp713.

[26] F. Moutier, A. Vandaele, and N. Gillis. Off-diagonal symmetric nonnegative matrix factorization. *Numerical Algorithms, pp*, pages 1–25, 2021.

[27] E. Mujuni and F. Rosamond. Parameterized complexity of the clique partition problem. In *Proceedings of the fourteenth symposium on Computing: the Australasian theory-Volume 77*, pages 75–78, 2008.

[28] National Heart, Lung, and Blood Institute (NHLBI. Marfan syndrome. https://www.nhlbi.nih.gov/health-topics/marfan-syndrome, (Accessed in March 4, 2021).

[29] Matthew R Nelson, Hannah Tipney, Jeffery L Painter, Judong Shen, Paola Nicoletti, Yufeng Shen, Aris Floratos, Pak Chung Sham, Mulin Jun Li, Junwen Wang, Lon R Cardon, John C Whittaker, and Philippe Sanseau. The support of human genetic evidence for approved drug indications. *Nat. Genet.*, 47(8):856–860, 2015.

[30] Milton Pividori, Nathan Schoettler, Dan L Nicolae, Carole Ober, and Hae Kyung Im. Shared and distinct genetic risk factors for childhood-onset and adult-onset asthma: genome-wide and transcriptome-wide studies. *Lancet Respir Med*, 7(6):509–522, 2019. PMCID: PMC6534440. doi:10.1016/S2213-2600(19)30055-4.

[31] Vivian Tam, Nikunj Patel, Michelle Turcotte, Yohan Bossé, Guillaume Paré, and David Meyre. Benefits and limitations of genome-wide association studies. *Nature Reviews Genetics*, 20(8):467–484, 2019. doi:10.1038/s41576-019-0127-1.

[32] Jaclyn N. Taroni, Peter C. Grayson, Qiwen Hu, Sean Eddy, Matthias Kretzler, Peter A. Merkel, and Casey S. Greene. Multiplier: A transfer learning framework for transcriptomics reveals systemic features of rare disease. *Cell Systems*, 8(5):380–394.e4, 2019. URL: http://www.sciencedirect.com/

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited

science/article/pii/S240547121930119X.

[33] P. M. Vaidya. Speeding-up linear programming using fast matrix multiplication. In *30th Annual Symposium on Foundations of Computer Science*, pages 332–337, 1989. doi:10.1109/SFCS.1989.63499.

[34] Kavitha Venkatesan, Jean-François Rual, Alexei Vazquez, Ulrich Stelzl, Irma Lemmens, Tomoko Hirozane-Kishikawa, Tong Hao, Martina Zenkner, Xiaofeng Xin, Kwang-Il Goh, Muhammed A. Yildirim, Nicolas Simonis, Kathrin Heinzmann, Fana Gebreab, Julie M. Sahalie, Sebiha Cevik, Christophe Simon, Anne-Sophie de Smet, Elizabeth Dann, Alex Smolyar, Arunachalam Vinayagam, Haiyuan Yu, David Szeto, Heather Borick, Amélie Dricot, Niels Klitgord, Ryan R. Murray, Chenwei Lin, Maciej Lalowski, Jan Timm, Kirstin Rau, Charles Boone, Pascal Braun, Michael E. Cusick, Frederick P. Roth, David E. Hill, Jan Tavernier, Erich E. Wanker, Albert-László Barabási, and Marc Vidal. An empirical framework for binary interactome mapping. *Nature Methods*, 6(1):83–90, 2009. doi:10.1038/nmeth.1280.

[35] Peter M Visscher, Naomi R Wray, Qian Zhang, Pamela Sklar, Mark I McCarthy, Matthew A Brown, and Jian Yang. 10 years of GWAS discovery: Biology, function, and translation. *Am. J. Hum. Genet.*, 101(1):5–22, 2017.

[36] Kyoko Watanabe, Sven Stringer, Oleksandr Frei, Maša Umićević Mirkov, Christiaan de Leeuw, Tinca J C Polderman, Sophie van der Sluis, Ole A Andreassen, Benjamin M Neale, and Danielle Posthuma. A global overview of pleiotropy and genetic architecture in complex traits. *Nat. Genet.*, 51(9):1339–1348, 2019.

[37] Sally E. Wenzel. Asthma phenotypes: the evolution from clinical to molecular approaches. *Nature Medicine*, 18(5):716–725, 2012. URL: https://doi.org/10.1038/nm.2678.

[38] Z. Y. Zhang, Y. Wang, and Y. Y. Ahn. Overlapping community detection in complex networks using symmetric binary matrix factorization. *Physical Review E*, 87:6, 2013.

Copyright © 2021 by SIAM
Unauthorized reproduction of this article is prohibited