Fundamental Limits of Multi-Sample Flow Graph Decomposition

Kayvon Mazooji*, Sreeram Kannan†, William Stafford Noble† and Ilan Shomorony*
*University of Illinois at Urbana-Champaign
†University of Washington

***Abstract*— The problem of decomposing a graph flow into a small set of paths has a wide range of applications, including transcriptome assembly and routing in data networks. A standard formulation is the sparsest flow decomposition problem, which is known to be NP-hard. In this work, we consider a multi-sample variant of this problem, motivated by the problem of identifying and quantifying proteoforms from mass spectrometry data, where multiple views of the graph can be obtained from multiple biological samples. We derive necessary conditions for the set of samples to unambiguously determine the ground truth set of paths, and we design an algorithm with matching sufficient conditions for a large class of problem instances, making our algorithm information optimal for this class of problem instances. The necessary conditions, combined with a probabilistic model for sample generation, yield a characterization of the number of samples needed for unambiguous recovery of the underlying paths. We analyze the algorithm's performance on flow data simulated on peptide graphs from real mass spectrometry data.**

I. Introduction

Twenty types of amino acids make up all the proteins found in the human body, and each protein is a sequence of 50–5000 amino acids. One method of identifying proteins and quantifying their abundances in a complex biological sample is through tandem mass spectrometry, where substrings (called peptides) of the proteins present are observed with varying abundances, and the proteins and their abundances are reconstructed from these peptides [1]. In addition to the mass spectrometry data, this process requires a database of proteins that are known to exist in the given sample. For example, for analysis of a human sample, we would use a database containing all proteins encoded in the human genome. Unfortunately, any such protein database is necessarily incomplete, because there inevitably exist protein variants in the sample that are not present in the database. We use the term *proteoform* to refer to variants of a protein that arise due to variation in how the protein is created (splice isoforms) and variants that arise after the protein is created (post-translational modifications) [2]. Using the observed peptides and their abundances in a sample, a directed graph can be created where peptides are nodes, and directed edges exist between adjacent peptides in known human proteins. If the abundances of the peptides are measured accurately enough, then a flow will form on the graph, and we can reconstruct the proteoforms and their abundances in the sample by decomposing the flow on the graph into weighted paths (see Figure 1). To aid the situation, multiple samples from the same tissue can be obtained, which contain the same set of proteoforms, providing us with multiple distinct flows on the same graph. Motivated by this, in this work we consider the problem of inferring the true set of paths that form multiple flows on a directed graph.

The problem of recovering the set of paths that produce an observed flow in a graph also has applications in the analysis of transcriptomic (i.e., gene expression) data [3–5], cancer genomic assembly [6], metagenomic assembly [7], virus quasi-species assembly [8], and routing in data networks [9]. A standard combinatorial optimization formulation of this problem is the *sparsest flow decomposition* problem [10]. In this problem, one observes a directed acyclic graph $G = (V, E)$ and a flow $w : E \to \mathbb{R}^+$ on its edges and seeks to recover the smallest number of paths with respective weights that explain the flow w (i.e., their weighted sum equals the flow). There are several difficulties with this approach. First, sparsest flow decomposition is in general NP-hard [9, 11]. Second, even if we could solve this problem efficiently, there is no guarantee that the recovered set of paths matches the true set of paths in a practical setting such as with RNA-seq data [3–5]. Finally, it is not clear how to define the sparsest flow decomposition when the flow conservation property does not hold exactly due to the addition of noise in the observed edge weights.

In this paper we consider an alternative information-theoretic formulation of the flow graph decomposition problem. For a known graph G, we assume the existence of an unknown set of paths P, where each path begins at a source and ends at a sink node in G. Motivated by the problem of identifying and quantifying the abundance of proteoforms from mass spectrometry data, where multiple samples can be obtained and are expected to contain the same set of proteoforms, we consider a multi-sample version of the problem. More precisely, we observe T flows on G, all of which are consistent with the set of paths P. Each flow is produced by assigning a non-negative weight to each path in P and assigning edge e a weight equal to the sum of the weights of the paths in P that pass through it (see Figure 1). We refer to each set of edge weight assignments as a *sample*. Given T samples of G, our goal is to recover the paths in P and their weights for each sample.

Unlike in the sparsest flow decomposition problem, in our formulation we can ask an information-theoretic question: When is there enough information in the T samples to allow P to be unambiguously recovered? To answer this question, we establish a necessary condition on the samples to guarantee that the paths in P must be present in any recovery $\hat{P}$ consis-

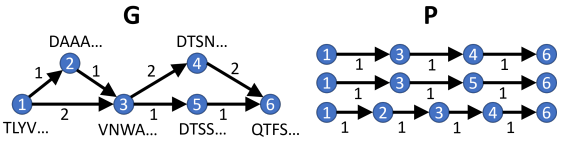

Fig. 1. Example of a peptide graph G from real mass spectrometry data. Each peptide node is associated with an amino acid sequence. In this example, the sparsest flow decomposition of G is not equal to the true set of paths P.

tent with the data. We then establish a sufficient condition on the samples and the graph G for correctly reconstructing all the paths in P using an algorithm we design. The necessary condition we derive is equivalent to the sufficient condition for a large class of (G, P), making our algorithm information-optimal for this class of (G, P). Under the assumption that the weights of paths in P are i.i.d. uniformly distributed, we derive an upper bound on the number of necessary and sufficient samples for this class of (G, P). We then test the algorithm's performance on flow data simulated on real peptide graphs obtained from mass spectrometry experiments.

II. Preliminaries and Problem Setting

Following standard convention, we use $[n] = \{1, 2, \ldots, n\}$. We let $G = (V, E)$ denote a directed acyclic graph with nodes V and edges E. An edge e that starts at node u and ends at node v is denoted by (u, v). We let $v_{\text{in}}(e)$ and $v_{\text{out}}(e)$ be the start and end nodes of e. For $v \in V$, an *in-edge* is an edge that ends at v, and an *out-edge* is an edge that begins at v. The set of in-edges for v is denoted by $E_{\text{in}}(v)$, and the set of out-edges for v is denoted by $E_{\text{out}}(v)$. The *in-degree* of v is equal to $|E_{\text{in}}(v)|$ and the *out-degree* is equal to $|E_{\text{out}}(v)|$. An in-edge of v begins at an *in-node* of v, and an out-edge of v ends at an *out-node* of v. The set of in-nodes of v is denoted by $V_{\text{in}}(v)$, and the set of out-nodes of v is denoted by $V_{\text{out}}(v)$. A *source* in G is a node with in-degree 0, and a *sink* in G is a node with out-degree 0. A *path* is a sequence of nodes $(v_1, v_2, ..., v_n)$ where (v_i, v_{i+1}) is an edge for $i \in [n-1]$. A *subpath* of a path p is a contiguous subsequence of nodes in p. With slight abuse of notation, for an edge $e \in E$ and path $p \in P$, we write $e \in p$ if $e = (v_i, v_{i+1})$ for adjacent nodes v_i and v_{i+1} in p. A *flow* on G is a mapping $w : E \to \mathbb{R}^+$ so that, $\forall\, v \in V$, $\sum_{e \in E_{\text{in}}(v)} w(e) = \sum_{e \in E_{\text{out}}(v)} w(e)$.

Let P denote the unknown set of paths in G. We observe T samples, each of which is a flow w_t, $t = 1, \ldots, T$. We denote the set of samples by $\mathcal{T}$. Each w_t must be *consistent* with the set of paths P; i.e., it must be obtained by assigning a non-negative abundance α_p to each path $p \in P$, and letting $w_t(e) = \sum_{p : e \in p} \alpha_p$. For a set of edges E', we let $w_t(E') = \sum_{e \in E'} w_t(e)$. For a subpath $(v_1, \ldots, v_k)$ of a path in P, we let $w_t((v_1, \ldots, v_k))$ denote the sum of the weights of all paths in P that include $(v_1, \ldots, v_k)$ as a subpath.

Given the set of observed flows $\mathcal{T}$, our goal is to recover the set of paths P. Rather than considering a combinatorial optimization formulation such as the sparsest flow decomposition, we instead ask an information-theoretic question: When does the set of samples $\mathcal{T}$ contain enough information for the unambiguous recovery of P? We formalize this as follows.

Definition 1. *A set of samples $\mathcal{T}$ for G reveals P if any set of paths P^0 consistent with $\mathcal{T}$ must satisfy $P \subset P^0$.*

Our goal is to find an algorithm that reconstructs P and the corresponding path weights for all samples from $\mathcal{T}$ whenever $\mathcal{T}$ reveals P. We are also interested in understanding how large $\mathcal{T}$ should be for it to reveal P. To study that question, we assume that the abundances $w_t(p)$ are i.i.d. $U(0, 1)$ random variables, for all $p \in P$, and we define:

Definition 2. *The sample complexity $T(\epsilon)$ is the minimum number of samples needed to guarantee that $\mathcal{T}$ reveals P with probability at least $1 - \epsilon$.*

The algorithms and theory discussed in the next sections will rely on the notion of decomposing a node $v \in V$ given T samples. A *decomposition* of v is the replacement of v in G by a set V^0 of nodes such that:

1) each $v^0 \in V^0$ has in-degree one and out-degree one, and has one in-node that is an in-node of v, and has one out-node that is an out-node of v;
2) the assignment of edge weights to each new edge in each sample is such that the modified graph still forms a flow for each sample.

In order to keep track of original paths in G when we decompose a node v into V^0, we need to define a function ℓ where $\ell(v^0) = v$ for all $v^0 \in V^0$. For a path $p = (v_1, v_2, \ldots, v_n)$ in G, we let $\ell(p) = (\ell(v_1), \ell(v_2), \ldots, \ell(v_n))$.

III. Main Results

Our first main result states that a simple algorithm for recovering P is information-optimal for a broad class of (G, P). To describe this class of (G, P), we will need the notion of a "good" node, which we define at the end of this section (Definition 3).

Theorem 1. *Suppose (G, P) is such that all nodes are good. Then the Topological Decomposition algorithm (Algorithm 1) outputs $\hat{P} = P$ with correct path weights for all samples if and only if the set of samples $\mathcal{T}$ reveals P.*

Theorem 1 implies that topologically decomposing nodes in G in the sparsest possible way (Algorithm 1) is information-optimal for the broad class of (G, P) such that all nodes are good. This is proved in Section IV by establishing necessary and sufficient conditions for $\mathcal{T}$ to reveal P for the class of (G, P) where all nodes are good.

Our second main result is an an upper bound on $T(\epsilon)$ for the class of (G, P) where all nodes are good.

Theorem 2. *Suppose (G, P) is such that all nodes are good. Let k be the maximum number of paths in P passing through any given node in G. Then*

$$\frac{\ln(1/\epsilon)}{-\ln(1 - \frac{1}{k!})} \leq T(\epsilon) \leq \frac{\ln \frac{|V|\,|E|}{\epsilon}}{-\ln\left(1 - \frac{1}{k!}\right)}. \tag{1}$$

Algorithm 1: Topological Decomposition

Data: $G = (V, E)$
Result: $\hat{P}$, $\{\hat{w}_t : t \in [T]\}$

Initiate a queue Q;
Add all nodes $v \in V$ with $|E_{\text{in}}(v)| > 1$ to Q in topological order;
while $Q \neq \emptyset$ **do**
  $v \leftarrow$ first node in Q;
  remove v from Q;
  decompose v into the smallest possible set V^0 ;
  for $u \in V_{out}(v^0)$ *for* $v^0 \in V^0$ **do**
    if $E_{in}(u) > 1$ *and* u *is not already in* Q **then**
      add u to the front of Q;
$\hat{P} \leftarrow$ the set of $(\ell(v_1), \ell(v_2), ..., \ell(v_k))$ for all paths $p = (v_1, v_2, ..., v_k)$ in decomposed graph that start at a source and end at a sink;
for p *in* $\hat{P}$ **do**
  $\hat{w}_t(p) \leftarrow$ weight of last edge in p in decomposed graph for sample t ;

Theorem 2 characterizes the sample complexity up to a log factor. We also point out that $-1/\ln(1 - 1/k!) \sim k!$. This shows that if the number of paths that pass through a single node is not too large, then the number of samples needed is reasonable for (G, P) where all nodes are good.

In order to formally define when a node is good, we first need to define, given G, P, and a node v, the *decomposed graph before* v, denoted by G'_v. The graph G'_v is obtained by decomposing every node before v in topological order in such a way that all paths in P are preserved and all nodes before v have in-degree at most one. This is illustrated in Figure 2(b). Observe that for each node u that comes before v in G'_v in topological order, there is only one path that starts at a source in G'_v and ends at u. We denote this unique path by $G'_v[: u]$. Any path $G'_v[: u]$ for u before v in topological order is a subpath of a path in P. Now, for a node v, we define a bipartite graph B_v as follows. The left nodes of B_v are the in-nodes of v in G'_v and the right nodes of B_v are the out-nodes of v in G'_v, and they are connected according to the paths P, as illustrated in Figure 3(c,d).

Definition 3. *A node v is good if B_v has no cycles.*

Notice that, if a node is not good, then the presence of a cycle in B_v allows an amount of flow ϵ to be added and subtracted in an alternating manner along the cycle while preserving flow conservation (Figure 2(e)). By focusing on (G, P) with only good nodes, we avoid this ambiguity and prove that Algorithm 1 is successful if $\mathcal{T}$ reveals P.

IV. Proof of Theorem 1

We prove Theorem 1 by first establishing (in Lemma 1) a necessary condition for $\mathcal{T}$ to reveal P that holds for any (G, P). This condition is based on an application of the

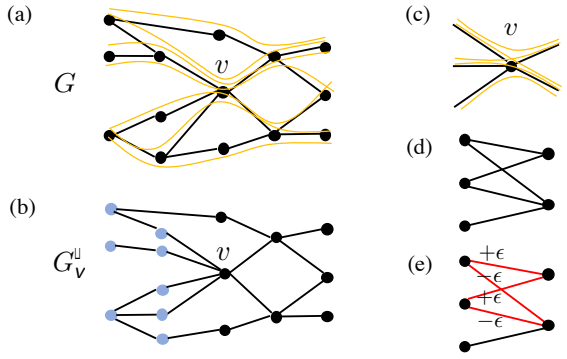

Fig. 2. (a) Example of a graph G and paths P. (b) For a node v, we build G'_v by decomposing all nodes before v in topological order (nodes in blue) so that the subgraph graph formed by these nodes is an arborescence. (c,d) The bipartite graph B_v is formed by the in-nodes and the out-nodes of v in G'_v, connected according to P. (d,e) If B_v contains a cycle, v is not good.

max-flow min-cut theorem [12]. For a node v, consider the decomposed graph before v, G'_v (see Figure 2(b)). Suppose $V_{\text{in}}(v) = \{x_1, \ldots, x_{jV_{\text{in}}(v)j}\}$ and $V_{\text{out}}(v) = \{y_1, \ldots, y_{jV_{\text{out}}(v)j}\}$. We then consider the graph H_v shown in Figure 3(a), obtained by first creating a complete bipartite graph between $V_{\text{in}}(v)$ and $V_{\text{out}}(v)$ (with capacity infinity for every edge) and then adding an in-edge to each x_i with capacity $w_t(x_i, v)$ and an out-edge to each y_i with capacity $w_t(v, y_i)$. Notice that any set of paths going through v satisfying the flow constraints in G'_v corresponds to a *maximum* flow in this graph (since the capacities of in-edges and out-edges will be fully used). Notice that the only two minimum cuts in H_v are $\{(s_i, x_i) : 1 \leq i \leq |V_{\text{in}}(v)|\}$ or $\{(y_i, t_i) : 1 \leq i \leq |V_{\text{out}}(v)|\}$, with min-cut value $w_t(E_{\text{in}}(v)) = w_t(E_{\text{out}}(v))$. This is because, if one in-edge, say (s_1, x_1), is not included in the cut, all out-edges must be included (or else an edge with infinite capacity will have to be picked).

Next, we notice that sample t reveals the existence of a path $p \in P$ going through (x_i, v, y_j) if and only if the removal of edge (x_i, y_j) in H_v reduces the min-cut value. To see this, we notice that, if the removal of (x_i, y_j) in H_v does not reduce the min-cut, by the max-flow min-cut theorem, there exists a flow in H_v that does not use (x_i, y_j) and still achieves total flow $w_t(E_{\text{in}}(v)) = w_t(E_{\text{out}}(v))$, which means that it is possible to have a flow decomposition P^0 of G'_v that does not include

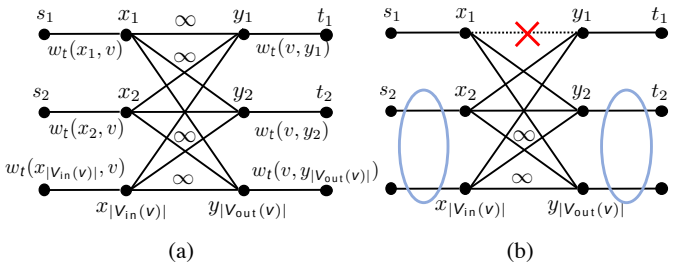

Fig. 3. (a) Graph H_v used to derive conditions for $\mathcal{T}$ to reveal P. Notice that any set of paths P consistent with $\mathcal{T}$ induces a maximum flow in G'_v with max-flow value $w_t(E_{\text{in}}(v)) = w_t(E_{\text{out}}(v))$. (b) By removing $(x_1; y_1)$, the only possible new minimum cut is the set of edges circled in blue.

the path p containing (x_i, v, y_j). Conversely, if the min-cut is reduced by the removal of (x_i, y_j), then any set of paths P that explains sample t must contain a path through (x_i, v, y_j).

Finally, we need to find conditions for the removal of (x_i, y_j) to reduce the min-cut value of H_v. Notice, from Figure 3(b), that in the graph obtained by removing (x_1, y_1) from H_v, the only possible minimum cuts are $\{(s_i, x_i) : 1 \leq i \leq |V_{\text{in}}(v)|\}$, $\{(y_i, t_i) : 1 \leq i \leq |V_{\text{out}}(v)|\}$, or

$$\{(s_i, x_i) : 2 \leq i \leq |V_{\text{in}}(v)|\} \cup \{(y_i, t_i) : 2 \leq i \leq |V_{\text{out}}(v)|\}.$$

This is because, any finite cut that does not contain (s_1, x_1) must contain $\{(y_i, t_i) : 2 \leq i \leq |V_{\text{out}}(v)|\}$ (and similarly for not containing (y_1, t_1)). Therefore, the removal of (x_1, y_1) reduces the min-cut of H_v if and only if

$$w_t(E_{\text{in}}(v) \backslash (x_1, v)) + w_t(E_{\text{out}}(v) \backslash (v, y_1)) < w_t(E_{\text{in}}(v)). \quad (2)$$

For a general edge (x_i, y_j), this condition can be equivalently expressed (by noticing that $w_t(E_{\text{in}}(v)) = w_t(E_{\text{out}}(v))$) as

$$w_t(E_{\text{in}}(v) \setminus (x_i, v)) < w_t((v, y_j)). \quad (3)$$

$$w_t(E_{\text{out}}(v) \setminus (v, y_j)) < w_t((x_i, v)), \quad (4)$$

Moreover, since

$$w_t(E_{\text{in}}(v) \setminus (x_i, v)) = \sum_{x \neq x_i} \sum_{y \neq y_j} w_t((x, v, y)) + w_t((v, y_j)) - w_t((x_i, v, y_j)),$$

condition (3) can be expressed in terms of triplets (x, v, y) as

$$\sum_{x \neq x_i} \sum_{y \neq y_j} w_t((x, v, y)) < w_t((x_i, v, y_j)). \quad (5)$$

Using the form in (3), the above discussion implies the following necessary conditions for $\mathcal{T}$ to reveal P:

Lemma 1. *In order for the set of samples $\mathcal{T}$ to reveal P, it must hold that, for all $v \in G$ and all $u, w \in G_v^0$ such that $(\ell(G_v^0[: u]), v, w)$ is a subpath of some path in P,*

$$w_t(E_{in}(v) \setminus (u, v)) < w_t((v, w)) \quad (6)$$

in G_v^0 for some sample $t \in [T]$.

Lemma 1 is illustrated in Figure 4. The two red paths in Figure 4(a) violate the condition in (6). From Lemma 1, $\mathcal{T}$ should not reveal their presence. This is illustrated in Figure 4(b), where a set of paths consistent with the flow can be found without including either of the red paths.

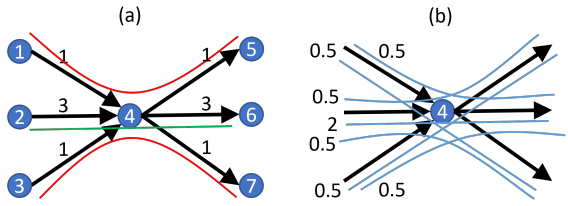

Fig. 4. (a) For the true paths $(1, 4, 5)$ and $(3, 4, 7)$ shown in red, the necessary condition is not satisfied. The necessary condition is satisfied for $(2, 4, 6)$, so it's present in every decomposition. (b) We can satisfy the flow constraint without including the paths $(1, 4, 5)$ or $(3, 4, 7)$ in the decomposition of 4.

In order to prove Theorem 1, we need to prove that Algorithm 1 recovers P correctly if, in addition to the necessary condition in Lemma 1, all nodes in G are good (Definition 3).

Lemma 2. *Algorithm 1 reconstructs P and the corresponding path weights for all T samples if $(G, \mathcal{T})$ satisfies the following:*

1) *all nodes in G are good.*
2) *for any $v \in G$, and any nodes u, w in G_v^0 such that $(\ell(G_v^0[: u]), v, w)$ is a subpath of a path in P,*

$$w_t(E_{in}(v) \setminus (u, v)) < w_t((v, w)) \quad (7)$$

in G_v^0 for some sample $t \in [T]$.

Proof: Consider a node v and let a be the node following v in topological order. We will prove that if G_v^0 is the graph produced by the algorithm before v is decomposed, and if conditions (1) and (2) hold for v, then G_a^0 is the graph after v is decomposed. Given this property, at the end of the algorithm, all nodes will be correctly decomposed with correct weights for each sample and the set of paths that start at a source and end at a sink in the resulting graph will be equal to P.

Suppose G_v^0 is the graph at the beginning of the iteration where v is decomposed. Suppose nodes $u, w \in G_v^0$ are such that $(\ell(G_v^0[: u]), v, w)$ is a subpath of a path in P. Let $e_1 = (u, v)$ and $e_2 = (v, w)$. If there is some $t \in [T]$ such that $w_t(E_{\text{in}}(v) \backslash e_1) < w_t(e_2)$, then clearly a path going through e_1 and e_2 must exist in P because there is not enough weight in the set of edges $E_{\text{in}}(v) \backslash e_1$ to completely satisfy the weight of e_2 in sample t. Therefore, if the second condition in the theorem statement is satisfied, all u, w such that $(\ell(G_v^0[: u]), v, w)$ is a subpath of a path in P are included in the decomposition of v, and there are no false connections between in-nodes and out-nodes of v in the decomposition since the algorithm picks the sparsest decomposition. This means that the decomposition of v corresponds exactly to the bipartite graph B_v used in Definition 3. Then, by the first condition, there is no cycle in B_v, which implies that there is a unique way to assign edge weights after decomposing v (since there are no cycles, one can repeatedly find a node with in- or out-degree one in B_v, assign the weight of the corresponding edge and remove it). Thus, we have recovered the true decomposition of v with the correct weights for each sample, and the invariant holds. ∎

V. Proof of Theorem 2

A. Upper Bound

We prove the upper bound on T (ϵ) by first calculating the probability that condition (2) in Lemma 2 holds in a sample. Given the equivalence between (4), (3) and (5), established in Section IV, we can instead compute the probability that, for some (u, v, w) that is a subpath of some $p \in P$, we have

$$\sum_{x \neq u} \sum_{y \neq w} w_t((x, v, y)) < w_t((u, v, w)), \quad (8)$$

which then implies that the tth sample reveals (x, y, z). In what follows, we let U denote a random variable uniformly

distributed in $(0,1)$ and let H_s denote a random variable with the Irwin-Hall distribution with parameter s.

Lemma 3. *Suppose that for some node v in G and nodes $u, w \in G_v^0$ there are a paths in P that go through the subpath $(\ell(G_v^0[:u]), v, w)$, and b paths that pass through v that do not go through either $\ell(G_v^0[:u])$ or w. The probability that (u,v,w) satisfies (8) is given by*

$$\frac{1}{(b+a)!} \sum_{n=0}^{b} (-1)^n \binom{b+a}{n} (a-n)^{b+a}. \tag{9}$$

Proof: We have that condition (8) holds with probability $P(H_b < H_a)$. The pdf of $H_b - H_a$ is given by the pdf of H_{b+a} shifted to the left by a. Thus,

$$\begin{aligned} P(H_b - H_a < 0) &= F_{H_{b+a}}(a) \\ &= \frac{1}{(b+a)!} \sum_{n=0}^{b} (-1)^n \binom{b+a}{n} (a-n)^{b+a} \end{aligned} \tag{10}$$

where F_X is the cumulative distribution function of X. ∎

With this lemma, we can prove Theorem 2. Recall that k is the maximum number of paths in P passing through any given node in G. Consider a node v in G. Then for a pair of nodes $u, w \in G_v^0$ such that $(\ell(G_v^0[:u]), v, w)$ is a subpath of a path in P, we have that (8) holds for a particular sample with probability at least $\frac{1}{k!}$ by assuming $a = 1$ and $b = k-1$ in Lemma 3. The probability that the condition does not hold for any sample is therefore at most $\left(1 - \frac{1}{k!}\right)^T$ by the independence of the samples. The probability that there is at least one $v \in G$ and $u, w \in G_v^0$ such that (8) does not hold for any sample is upper bounded by $|V| k \left(1 - \frac{1}{k!}\right)^T$ by the union bound since there are at most k such pairs of nodes $u, w \in G_v^0$ for each $v \in G$. Thus the probability the conditions in Theorem 1 hold is lower bounded by $1 - |V| k \left(1 - \frac{1}{k!}\right)^T$. For a probability of error at most ϵ, we solve for T to obtain the result.

B. Lower Bound

Suppose the graph is as follows. The set of nodes is $V = \{u_1, u_2, \ldots, u_k\} \cup \{v\} \cup \{x_1, x_2, \ldots, x_k\}$. The set of edges is $E = \{(u_1, v), (u_2, v), \ldots, (u_k, v)\} \cup \{(v, x_1), (v, x_2), \ldots, (v, x_k)\}$. The set of paths is $P = \{(u_1, v, x_1), (u_2, v, x_2), \ldots, (u_k, v, x_k)\}$. Clearly, all nodes are good for this problem instance. In order to reveal path (u_1, v, x_1), we must have $w_t(E_{\text{in}}(v) \setminus (u_1, v)) < w_t((v, x_1))$ for some sample t by Theorem 1. This occurs in a sample with probability $\frac{1}{k!}$ by Lemma 3. The probability this doesn't occur in T samples is given by $(1 - \frac{1}{k!})^T$. Solving for T, we find that a lower bound on $T\ (\epsilon)$ for this problem instance is

$$T\ (\epsilon) \geq \frac{\ln(\epsilon)}{\ln(1 - \frac{1}{k!})} \sim k! \ln(1/\epsilon). \tag{11}$$

VI. Faster Algorithm for Decomposing a Node

In Algorithm 1, for each v with more than one in-edge, we find the sparsest decomposition of the node v. Finding the sparsest decomposition of v using brute force search quickly becomes computationally infeasible as the number of in-edges and out-edges of v increases. We therefore use Algorithm 2 in simulation for fast node decomposition. Theorem 1 still holds when Algorithm 2 is used in place of sparsest node decomposition in Algorithm 1.

Algorithm 2 relies on the following optimization problem. Given a set of new nodes and new edges that may or may not decompose v in G, the optimization problem finds the edge weights that minimize the sum of squared violations of the flow constraints. For a set S of node pairs u, w where $u \in V_{\text{in}}(v)$ and $w \in V_{\text{out}}(v)$, let $s[0] = u$ and $s[1] = w$. The quadratic program is given by

$$\begin{aligned} \min_{z_{s;t}\ 0 \leq s \leq S;\ t \in [T]} \sum_{t=1}^{T} \Bigg(& \sum_{u \in V_{\text{in}}(v)} \Big(w_t((u,v)) - \sum_{s:\, s[0]=u} z_{s;t} \Big)^2 \\ & + \sum_{w \in V_{\text{out}}(v)} \Big(w_t((v,w)) - \sum_{s:\, s[1]=w} z_{s;t} \Big)^2 \Bigg). \end{aligned} \tag{12}$$

where the variable $z_{s;t}$ represents the weight of the path going from $s[0]$ to $s[1]$ that we choose. Let $f(G, v, S)$ be the optimal value of the program and $g(G, v, S)$ be the optimal solution.

Algorithm 2: Fast Node Decomposition

```
Data: G, v, ε
Result: S, q
S ← {{u, w} : (u, v, w) must exist in decomposition
  of v} (computed using condition (3)) ;
b ← f(G, v, S);
q ← g(G, v, S);
while b > ε do
    R ← ∅;
    for {u, w} ∉ S do
        S⁰ ← S ∪ {{u, w}};
        b⁰ ← f(G, v, S⁰);
        if b⁰ < b then
            b ← b⁰;
            R ← S⁰;
            q ← g(G, v, S⁰);
    S ← R;
```

VII. Simulations

We now discuss an example of our theory applied to a proteoform inference problem. We construct a peptide graph from a proteoform database as follows.

1) For each proteoform in the database of known proteoforms, add all of its peptides as nodes in the graph.
2) Connect these observed peptide nodes with directed edges reflecting the order in which they appear in the proteoform.
3) Set the weight of a peptide node equal to the peptide's observed abundance.

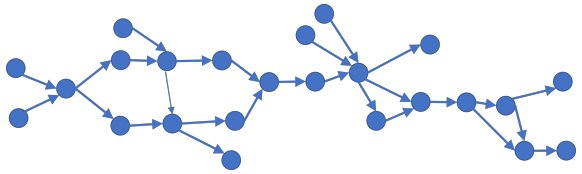

Fig. 5. This is the structure of the peptide graph used for the simulations. Each path in the original graph containing only nodes with one in-edge and one out-edge was condensed into one node.

If possible, we calculate the unique set of edge weights in the graph that are consistent with the observed node weights. We then want to infer the true paths in the graph given the samples that produced distinct sets of edge weights, including paths that were not used to construct the peptide graph.

In the simulation, we used a true peptide graph constructed as above, but did not have access to the ground truth proteoforms for a real data set. We therefore let P equal the set of proteoforms used to construct the peptide graph, and let the weight of each path in P be i.i.d. uniformly distributed between 0 and 1 fora each sample as Section IV.

The graph we used (Figure 5) contains seven proteoform paths ($|P| = 7$) and 72 peptide nodes ($|V| = 72$). There is a unique edge weight assignment given the node weights for this graph. Also, all nodes are good in this graph. We used 100 trials to estimate the mean number of samples necessary for the condition in Theorem 1 to hold and obtained 39.08.

For a range of number of samples T, we ran Algorithm 1 using the fast algorithm we designed for node decomposition in Section VI, and recorded the number of paths in P that were reconstructed and the total number of paths output. We ran five trials for each number of samples, and recorded the averages of the statistics for each number of samples (see Figure 6). We observe that the algorithm was performing perfectly even before the the average number of samples for the Theorem 1 to hold. We also added i.i.d. Gaussian noise with mean $\mu = 0$ and standard deviation $\sigma = 0.025$ to the the edge weights and ran the algorithm (see Figure 6). We observe that the noise did not affect the algorithm's performance much at this noise level.

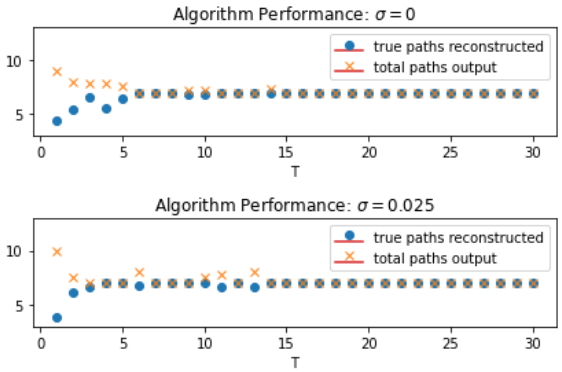

Fig. 6. The top plot shows the performance of the algorithm when no noise is added to the edges, and the bottom plot shows the performance when independent Gaussian noise with standard deviation = 0:025 was added to each edge weight.

References

[1] H. Steen and M. Mann, "The ABC's (and XYZ's) of peptide sequencing," *Nature Reviews Molecular Cell Biology*, vol. 5, pp. 699–711, 2004.

[2] L. M. Smith and N. L. Kelleher, "Proteoform: a single term describing protein complexity," *Nature Methods*, vol. 10, pp. 186–187, 2013.

[3] Z. Wang, M. Gerstein, and M. Snyder, "Rna-seq: A revolutionary tool for transcriptomics," *Nature reviews. Genetics*, vol. 10, pp. 57–63, 12 2008.

[4] S. Kannan, J. Hui, K. Mazooji, L. Pachter, and D. Tse, "Shannon: An information-optimal de novo rna-seq assembler," *bioRxiv*, 2016.

[5] S. Mao, L. Pachter, D. Tse, and S. Kannan, "Refshannon: A genome-guided transcriptome assembler using sparse flow decomposition," *PLOS ONE*, vol. 15, pp. 1–14, 06 2020.

[6] V. Bansal and V. Bafna, "Hapcut: An efficient and accurate algorithm for the haplotype assembly problem," *Bioinformatics (Oxford, England)*, vol. 24, pp. i153–9, 09 2008.

[7] A. Afiahayati, K. Sato, and Y. Sakakibara, "Metavelvet-sl: An extension of the velvet assembler to a de novo metagenomic assembler utilizing supervised learning," *DNA research : an international journal for rapid publication of reports on genes and genomes*, vol. 22, 11 2014.

[8] A. Töpfer, T. Marschall, R. A. Bull, F. Luciani, A. Schönhuth, and N. Beerenwinkel, "Viral quasispecies assembly via maximal clique enumeration," *PLOS Computational Biology*, vol. 10, pp. 1–10, 03 2014.

[9] T. Hartman, A. Hassidim, H. Kaplan, D. Raz, and M. Segalov, "How to split a flow?," in *2012 Proceedings IEEE INFOCOM*, pp. 828–836, 2012.

[10] M. Shao and C. Kingsford, "Theory and a heuristic for the minimum path flow decomposition problem," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 16, no. 2, pp. 658–670, 2019.

[11] B. Vatinlen, F. Chauvet, P. Chrétienne, and P. Mahey, "Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths," *European Journal of Operational Research*, vol. 185, pp. 1390–1401, 03 2008.

[12] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian journal of Mathematics*, vol. 8, pp. 399–404, 1956.