Flow Decomposition with Subpath Constraints

Lucia Williams
School of Computing, Montana State University, Bozeman, MT, USA

Alexandru I. Tomescu
Department of Computer Science, University of Helsinki, Finland

Brendan Mumey
School of Computing, Montana State University, Bozeman, MT, USA

Abstract

Flow network decomposition is a natural model for problems where we are given a flow network arising from superimposing a set of weighted paths and would like to recover the underlying data, i.e., *decompose* the flow into the original paths and their weights. Thus, variations on flow decomposition are often used as subroutines in multiassembly problems such as RNA transcript assembly. In practice, we frequently have access to information beyond flow values in the form of *subpaths*, and many tools incorporate these heuristically. But despite acknowledging their utility in practice, previous work has not formally addressed the effect of subpath constraints on the accuracy of flow network decomposition approaches. We formalize the *flow decomposition with subpath constraints* problem, give the first algorithms for it, and study its usefulness for recovering ground truth decompositions. For finding a minimum decomposition, we propose both a heuristic and an FPT algorithm. Experiments on RNA transcript datasets show that for instances with larger solution path sets, the addition of subpath constraints finds 13% more ground truth solutions when minimal decompositions are found exactly, and 30% more ground truth solutions when minimal decompositions are found heuristically.

2012 ACM Subject Classification Theory of computation → Network flows; Applied computing → Computational transcriptomics

Keywords and phrases Flow decomposition, subpath constraints, RNA-Seq

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.16

Supplementary Material *Software (Source Code)*: `https://github.com/msu-alglab/coaster`

Funding This work was partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 851093, SAFEBIO), the Academy of Finland (grants No. 322595, 328877), the US Fulbright program, the Fulbright Finland Foundation, the Helsinki Institute for Information Technology (HIIT), as well as the US National Science Foundation (grant DBI-1759522).

Acknowledgements Computational efforts were performed on the Hyalite High Performance Computing System, operated and supported by University Information Technology Research Cyberinfrastructure at Montana State University.

1 Introduction

Flow networks are useful models in many domains, from transportation planning to computational biology. In some cases, the flow on a graph arises as the superposition of some set of weighted paths, such as trips through a road network, routing of information through a communication network, or paths in a graph encoding mixed reads sequenced from several biological sequences, as in the case of RNA transcripts through a splice graph.

In many such applications, we are actually presented with the inverse problem: given a flow in a graph, we need to recover the initial paths that made up the flow. This problem is also referred to as the *flow decomposition (FD) problem.* In computational biology, this is a common subroutine in multiassembly problems, such as RNA transcript assembly or viral quasispecies assembly. Prioritizing parsimonious solutions proved to be an accurate

© Lucia Williams, Alexandru I. Tomescu, and Brendan Mumey;
licensed under Creative Commons License CC-BY 4.0

assembly method, but it can suffer when there are multiple parsimonious solution to choose from. As such, in this paper we consider a natural generalization of the flow decomposition problem, by assuming that extra information about the initial paths is available in the form of *subpath constraints*. These are paths in the network graph that must be followed by at least one path in the flow decomposition; thus, we are looking for flow decompositions with the property that every constraint is a subpath of some decomposition path. We call the resulting problem *flow decomposition with subpath constraints (FDSC)*.

Biological setting. Algorithms that solve variations of the flow decomposition problem are at the heart of most RNA transcript assembly software, including IsoLasso [16], Traph [25], FlipFlop [9], Scallop [21] and StringTie [18]. More recently, flow decomposition methods were used for another multi-assembly problem, namely strain-aware genome assembly, with applications to viral quasispecies assembly [4, 2]. Briefly, flow decomposition methods for sequence assembly work by using reads and their abundances to first construct a flow network whose vertices may represent exons (in the case of an RNA splice graph) or k-mers (in the case of a de Bruijn graph). Edges in the network are present if there is read evidence that some sequence followed the edge (e.g. two exons are consecutive in some transcript). Furthermore, each edge is weighted by the number of reads that support it. With perfect data, we might expect the weights to directly provide a flow in the network; however in practice some adjustment to the weights may be needed to achieve a flow. One such method uses a minimum-cost flow approach for this adjustment [25]. Another approach [28] models the input as an *inexact flow* network in which edge flows belong to intervals, that are estimated from the data. In all cases, we seek a path decomposition for the flow network that minimizes the number of paths.

In Kloster et al. [13] it was shown that in the case of RNA transcripts, most of the time the "true" transcripts also provide a minimum flow decomposition of the splice graph. However, there can often be more than one solution to the minimum flow decomposition problem; indeed, Kloster et al. found that, when the number of true transcripts is seven, the minimum flow decomposition found corresponds to the true paths in only 80% of the instances of that size, with lower accuracies as the number of true paths increases. In fact, practical methods for RNA assembly methods also have a precision of 50%-60% on some human datasets [23, 18]. Adding subpath constraints to the flow decomposition problem may further restrict the solution space, thus improving the RNA assembly accuracy.

In practice, the subpath constraints can be derived from reads overlapping three or more nodes of the flow graph. Long RNA-Seq reads naturally have this property in many cases; however, also short reads can exhibit this behavior in the case of short exons. As we review below, other possible sources of such constraints exist in practice as well, such as from partial assemblies, or *super-reads* [18] constructed from short reads that can be uniquely extended.

Finally, most of the RNA assembly tools cited above work in a so-called *genome-guided* setting in which also a reference genome of the studied species is available. This makes the splice graph acyclic (i.e. a *DAG*). While both the original flow decomposition problem and our variant with subpath constraints can be defined in flow networks with cycles (which would correspond to a *de novo assembly* setting), in this paper we focus on DAGs only.

Related work. Finding a flow decomposition with the minimum number of weighted paths is a well-studied problem in computer science. Even when restricted to DAGs, the minimum FD problem is NP-hard [27], and thus various practical approaches to it exist: approximation algorithms [11, 24, 19, 17, 5, 6], FPT algorithms [13], greedy algorithms [27, 22]. By taking the set of subpaths constraints to be empty (or to correspond to all edges of the graph with non-zero flow), it follows that also finding a solution to the FDSC problem with a *minimum* number of paths is NP-hard.

The idea of improving RNA assembly by multi-edge subpath information is in fact used by several flow-based tools, such as [21] and StringTie [18]. However, both approaches integrate subpaths in a heuristic manner, with no overall formulation of the computational problem they are solving. The same holds also for the viral quasispecies assembler [4]. Recently, the method TransBorrow [29] uses partial assemblies from different RNA assembly tools, and works by heuristically extending the subpaths they correspond to in a splice graph.

Moreover, our FDSC problem generalizes a related problem on DAGs. Recall that in the *minimum path cover (MPC)* problem, we are looking for a minimum-cardinality set of path that together cover all nodes of a DAG (e.g. "explain" all exons of a splice graph). The problem is behind early RNA assembly methods such as Cufflinks [26], and early virus quasispecies assembly methods such as ShoRAH [30]. The MPC problem has been extended to include subpath constraints as well [20, 15, 8, 14], by analogously requiring that each constraint is a subpath of some solution path. While these generalizations are polynomially-time solvable, they (together with the initial MPC formulations) are usually unsatisfactory since they ignore the weights of the graph (i.e. the abundances of the reads) – recall that most state-of-the-art RNA assembly methods cited above are flow-based. Moreover, MPCs and MPCs with subpath constraints correspond to restricted classes of flows in some DAG [7, 20], and thus the minimum FDSC problem is a strict generalization of the MPC problem with subpath constraints.

Contributions. In this work, we initiate the formal study of the FDSC problem. This is a natural model for multiassembly problems, as seen by the abundance of methods and tools that incorporate subpath information for improving RNA and viral quasispecies assemblies. However, because finding a *minimum* solution to the FDSC problem is NP-hard, these methods and tools have focused on either heuristic approaches or a polynomial-time solvable particular version of the problem (MPC) that ignores valuable edge weight information. Here, we make two advances that bring us closer to being able to use the complete version of the problem in practical tools. On the theoretical side, we formalize the problem and give the first algorithm to determine whether an instance is feasible (Theorem 17), and produce a solution if it is. The algorithm works via a reduction to the standard flow decomposition problem where any solution must translate to a solution in the original graph that satisfies all of the subpath constraints. Additionally, we give an FPT algorithm for the minimum FDSC problem (Theorem 19), extending the one of Kloster et al. [13].

We implement both of these algorithms, and perform a proof-of-concept study of their usefulness in RNA assembly. We experiment on a dataset developed by Shao et al. [22] to study their heuristic for the minimum FD problem. The same dataset was then used by Kloster et al. [13], who focused on studying the usefulness of standard minimum flow decompositions in RNA assembly, as explained above. We find that our FDSC algorithms increase our ability to uncover the ground truth RNA transcripts, as more and longer subpath constraints are included in the input. This holds both when minimality is enforced, through our FPT algorithm, and when it is only heuristically sought, through the flow decomposition reduction and an associated path reduction heuristic. For example, when there are seven ground truth transcripts, we increase the accuracy by 13% when an optimal solution is found (via FPT) and 30% when a heuristic solution is found.

When no edge appears in more subpath constraints than its flow value, our FDSC algorithm runs in polynomial time (i.e. always finds a solution to the FDSC problem, not necessarily minimum). Though we desire a minimal such path decomposition, algorithms that guarantee such solutions in general may be too slow to be used in practice. Despite a lack of minimality guarantees in our heuristic FDSC algorithm, our experiments show that the addition of subpath constraints yields solutions that approach the accuracy of a minimum decomposition without subpath constraints; thus, our results show that heuristic FDSC is a practical substitute for minimum FD without subpath constraints. On the other hand, when for some edges the number of subpath constraints exceed its flow value, our algorithm takes exponential time even to decide whether an FDSC solution exists (not necessarily minimum). We leave open the complexity of the FDSC problem (e.g. bring it in P, or classify it as NP-complete). This is an interesting question especially since finding one standard flow decomposition can be done in polynomial time [1]. Since flow decompositions are basic concepts in flow networks with a large body of research, we believe that also its natural generalization as the FDSC problem may lead to further developments in network flow theory and in its various applications.

2 Preliminaries

Since flow weights represent read counts, we restrict attention to integral flow networks and flow decompositions.

▶ **Definition 1.** *A* flow network $G = (V, E, f)$ *is a directed acyclic graph (DAG) comprised of a set of vertices V containing a source s and sink t, a set of directed edges E, and a flow function $f : E \to \mathbb{N}$, such that for $v \in V \setminus \{s, t\}$,*

$$\sum_{u:(u,v)\in E} f(u, v) = \sum_{w:(v,w)\in E} f(v, w). \tag{1}$$

Finally, for each $v \in V$, there is an s-t path in G that includes v.

▶ **Definition 2** (Flow decomposition)**.** *A* flow decomposition *for a flow network $G = (V, E, f)$ consists of set of s-t paths $\mathcal{P} = (P_1, \ldots, P_k)$ and associated integral flow weights, $w = (w_1, \ldots, w_k)$ with $w_i \in \mathbb{N}$ such that for each edge $e \in E$,*

$$\sum_{i:e\in P_i} w_i = f(e). \tag{2}$$

We define several problems concerning finding decompositions of flow networks into paths.

▶ **Problem 3** (MFD)**.** *Given a flow network $G = (V, E, f)$, the* minimum flow decomposition problem *is to find a decomposition $(\mathcal{P}, w)$ such that $|\mathcal{P}|$ is minimized.*

▶ **Definition 4** (Flow decomposition with subpath constraints)**.** *Let $G = (V, E, f)$ be a flow network.* Subpath constraints *are defined to be a set of simple paths $\mathcal{R} = \{R_1, \ldots, R_\ell\}$ in G such that no $R_i \subseteq R_j$. A flow decomposition $(\mathcal{P}, w)$ satisfies the subpath constraints if and only if*

$$\forall R_i \in \mathcal{R}\ \exists P_j \in \mathcal{P} \text{ such that } R_i \text{ is a subpath of } P_j \text{ (in short, } R_i \in P_j\text{)}. \tag{3}$$

Figure 1 shows an example of a flow network with subpath constraints.

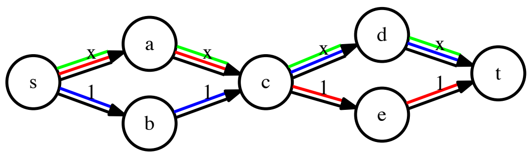


Figure 1 An example FDSC flow network with the flow values of the edges being 1 or x; the colored paths indicate the subpath constraints. If $x = 1$, then the instance is infeasible. If $x = 2$, then the instance is feasible and requires three paths to decompose (whereas the associated FD instance without subpath constraints can be decomposed with two paths in both cases).

▶ **Problem 5** (FDSC). *Given a flow network $G = (V, E, f)$ and subpath constraints $\mathcal{R}$, the* flow decomposition with subpath constraints *problem is to determine if there exists, and if so, find a flow decomposition $(\mathcal{P}, w)$ satisfying (3).*

▶ **Problem 6** (MFDSC). *Given a flow network $G = (V, E, f)$ and subpath constraints $\mathcal{R}$, the* minimum flow decomposition with subpath constraints *problem is to determine if there exists, and if so, find a flow decomposition $(\mathcal{P}, w)$ satisfying (3) such that $|\mathcal{P}|$ is minimized.*

3 Algorithms

3.1 FDSC reduces to FD

We now describe a reduction from the FDSC problem to the FD problem. The idea is to convert subpath constraints into edges in an FD instance so that any path decomposition of the FD instance translates into a path decomposition for the FDSC instance that covers the subpath constraints.

Given a flow network $G = (V, E, f)$ with subpath constraints $\mathcal{R}$, we define the *overdemand* of an edge as

$$d_o(e) = \max(0, |\{i : e \in R_i\}| - f(e)), \tag{4}$$

and say that e is *overdemanded* if $d_o(e) > 0$. The FDSC problem $(G, \mathcal{R})$ may be feasible if multiple subpaths covering e are satisfied by a single path in a path decomposition. If no edges are overdemanded, we can give a simple reduction from FDSC to FD by transforming all subpath constraints in the FDSC instance into edges in the FD instance.

▶ **Lemma 7.** *Let $G = (V, E, f)$ be a flow network with subpath constraints $\mathcal{R}$ such that no edge is overdemanded. Let $G' = (V, E', f')$ be the flow network that results from converting every subpath constraint $R_i = [v_1, v_2, \ldots, v_{|R_i|}]$ to a* bridge edge *$e_i = (v_1, v_{|R_i|})$ with $f'(e_i) = 1$ and subtracting one from the flow values on the edges it covers. That is, for all $e \in E$, $f'(e) = f(e) - |\{i : e \in R_i\}|$. G' is a flow network.*

Proof. Consider building G' from G iteratively by converting each subpath constraint into a new edge and subtracting its flow from the edges it covers. At each step, conservation of flow holds. Thus, after the final step, f' is a flow on G'. Additionally, because no edge is overdemanded, all flow values in f' are nonnegative. Thus, G' is a flow network. ◀

▶ **Lemma 8.** *Let G and G' be as described in Lemma 7. Let $(\mathcal{P}', w)$ be a size k solution to the FD problem on G'. There exists a size k solution to the FDSC problem on G.*

Proof. We show how to construct a size k solution to FDSC on G from $(\mathcal{P}', w)$. For each path $P' \in \mathcal{P}'$, create a new path P by replacing all bridge edges e_i' with the original sequence of nodes R_i. By construction, R_i must form a path from the start node of e_i to the end node of e_i in P, and so P is a valid path from s to t in G. We take $\mathcal{P}$ to be the set of all k such paths P. We now must show that $(\mathcal{P}, w)$ forms a flow decomposition with subpath constraints for G.

Let e be any edge in G and let $\mathcal{R}' \subseteq \mathcal{R}$ be the set of subpath constraints containing e. We can divide the paths in $\mathcal{P}$ that cover e into two disjoint sets: $\mathcal{P}_B$, those that covered bridge edges $e_i : R_i \in \mathcal{R}'$, and $\mathcal{P}_O$, those those that covered the original edge e in G'. Because $(\mathcal{P}', w)$ is a flow decomposition for G', each path in $\mathcal{P}_B$ must have unit weight. Thus, paths in $\mathcal{P}_B$ contribute $|\{i : R_i \in \mathcal{R}'\}|$ to e's flow. On the other hand, paths in $\mathcal{P}_O$ must cover e's flow in G', which is $f(e) - |\{i : R_i \in \mathcal{R}'\}|$. Thus, paths from $\mathcal{P}_B$ and $\mathcal{P}_O$ together cover e with exactly $f(e)$ units of flow. Additionally, $\mathcal{P}_B$ must satisfy all of the subpath constraints $\mathcal{R}'$, so together $\mathcal{P}_B$ and $\mathcal{P}_O$ do as well. ◀

Because any FDSC instance without overdemanded edges can be solved by reduction to FD, it follows that all FDSC instances without overdemanded edges are feasible.

▶ **Corollary 9.** *Let $G = (V, E, f)$ be a flow network with subpath constraints $\mathcal{R}$. A sufficient condition for a flow decomposition to exist is that no edge is overdemanded.*

When an FDSC instance has an overdemanded edge, the reduction given above fails, because any overdemanded edge would have a negative flow value after subtracting all of its demands from its original flow. However, if the FDSC instance $(G, \mathcal{R})$ is feasible, it is possible to first transform $(G, \mathcal{R})$ to an FDSC instance $(G, \mathcal{R}^*)$, where no edge is overdemanded and any path decomposition for $(G, \mathcal{R}^*)$ also provides a feasible path decomposition for $(G, \mathcal{R})$. By Lemma 7, $(G, \mathcal{R}^*)$ can be solved via reduction to an FD instance.

▶ **Lemma 10.** *Let $(G, \mathcal{R})$ be a feasible FDSC instance with overdemanded edge e and $(\mathcal{P}, w)$ be a path decomposition for $(G, \mathcal{R})$. Let $\mathcal{R}' \subseteq \mathcal{R}$ be the set of subpath constraints that contain e. There must be some $P \in \mathcal{P}$ such that $|\{R_i : R_i \in \mathcal{R}', R_i \in P\}| \geq 2$.*

Proof. Suppose not. That is, suppose $(\mathcal{P}, w)$ is a path decomposition for $(G, \mathcal{R})$ but no path in $\mathcal{P}$ covers two or more subpath constraints in $\mathcal{R}'$ completely. This means that every subpath constraint in $\mathcal{R}'$ must be satisfied by a different path; call this set of paths $\mathcal{P}'$ and let the total weight assigned to these paths be $w' \geq |\mathcal{P}'| = |\mathcal{R}'| = |\{i : e \in R_i\}|$. As e is overdemanded, we have $|\{i : e \in R_i\}| > f(e)$. But then $w' > f(e)$, contradicting the fact that $(\mathcal{P}, w)$ is a path decomposition for $(G, \mathcal{R})$. ◀

▶ **Definition 11** (Compatible Subpaths). *Two subpaths $R_i, R_j \in \mathcal{R}$ are* compatible *if and only if R_i and R_j have a suffix-prefix overlap (so that $R_i \cup R_j$ forms a simple path in G).*

▶ **Definition 12** (Directly Compatible Subpaths). *Two subpaths $R_i, R_j \in \mathcal{R}$ are* directly compatible *if and only if R_i and R_j are compatible and there does not exist a subpath R_k such that R_i and R_k are compatible and R_k and R_j are compatible.*

We remark that the directly compatible relation is just the transitive reduction of the compatible relation.

▶ **Lemma 13.** *Let $(G, \mathcal{R})$ be an FDSC instance with some overdemanded edge e. Then $(\mathcal{P}, w)$ is a solution for $(G, \mathcal{R})$ if and only if there exist directly compatible subpaths R_i and R_j, each containing e, such that $(\mathcal{P}, w)$ is a solution for $(G, \mathcal{R} \cup \{R_i \cup R_j\} \setminus \{R_i, R_j\})$.*

Proof. ($\rightarrow$) Let $(G, \mathcal{R})$ be a feasible FDSC instance with overdemanded edge e. Let $(\mathcal{P}, w)$ be a path decomposition for $(G, \mathcal{R})$. Let $\mathcal{R}' \subseteq \mathcal{R}$ be the set of subpath constraints that contain e. By Lemma 10, there exists a $P \in \mathcal{P}$ and $R_i, R_j \in \mathcal{R}'$ such that $R_i \neq R_j$ and $R_i, R_j \in P$. Since R_i and R_j both belong to P and overlap (since they each contain e), it follows that they are compatible. If R_i and R_j are not directly compatible, there must exist some R_k such that R_i and R_k both contain e and are directly compatible. In this case, take R_j to be R_k. Furthermore, the path P satisfies the subpath constraint $R_i \cup R_j$, so $(\mathcal{P}, w)$ is a feasible solution for $(\mathcal{G}, \mathcal{R} \cup \{R_i \cup R_j\} \setminus \{R_i, R_j\})$.

($\leftarrow$) Let R_i and R_j be directly compatible subpaths that both contain e. Let $(\mathcal{P}, w)$ be a feasible solution to $(\mathcal{G}, \mathcal{R} \cup \{R_i \cup R_j\} \setminus \{R_i, R_j\})$. It follows that there exists a path $P \in \mathcal{P}$ that covers $R_i \cup R_j$. Clearly, P also covers R_i and R_j, so $(\mathcal{P}, w)$ is also a feasible solution for $(G, \mathcal{R})$. ◀

▶ **Corollary 14.** *Let $(G, \mathcal{R})$ be an FDSC instance. If there are no compatible subpaths R_i and R_j containing some overdemanded edge e, then $(G, \mathcal{R})$ is infeasible.*

▶ **Definition 15** (Total Overdemand). *Let $(G, \mathcal{R})$ be an FDSC instance. The* total overdemand *is defined as*

$$D_o(G, \mathcal{R}) = \sum_{e \in E} d_o(e). \tag{5}$$

▶ **Lemma 16.** *Let $(G, \mathcal{R})$ be an FDSC instance with directly compatible constraints subpaths R_i and R_j, each containing some overdemanded edge e. Then, $D_o(\mathcal{G}, \mathcal{R} \cup \{R_i \cup R_j\} \setminus \{R_i, R_j\}) < D_o(G, \mathcal{R})$.*

Proof. All overdemanded edges in $R_i \cap R_j$ have their overdemand decreased by one. Since $e \in R_i \cap R_j$, there is at least one such edge. The overdemand of all remaining edges is unchanged. ◀

▶ **Theorem 17.** *Let $(G, \mathcal{R})$ be an FDSC instance with $|\mathcal{R}| = \ell$ and at least one overdemanded edge. In $O(\ell^{2 \cdot \min(D_o(G, \mathcal{R}), \ell)}|E| + \ell^3)$ time, we can determine whether $(G, \mathcal{R})$ is feasible, and if so, reduce $(G, \mathcal{R})$ to an FD instance on a modified network with at most ℓ additional edges.*

Proof. Lemma 13 suggests a strategy for processing FDSC instances with overdemanded edges: pick an overdemanded edge e and recursively test each possible union of directly compatible pairs of subpath constraints that contain e. If there are no compatible subpath constraints for some overdemanded edge, then by Corollary 14, that branch of the recursion cannot lead to a feasible solution. On the other hand, if we reach an FDSC instance with no overdemanded edges, then we can solve this instance using the reduction to FD described in Lemma 7.

The branches of this recursive algorithm form a tree with maximum branching factor $\binom{\ell}{2}$ (since we must try each pair of directly compatible constraint subpaths containing an overdemanded edge). By Lemma 16, the total overdemand is reduced by at least one at one at each new level in the recursion tree. We also note that the number of subpath constraints is reduced by one at each new level. It follows that the maximum depth of recursion is at most $\min(D_o(G, \mathcal{R}), \ell)$. The non-recursive work can be made to take $O(|E|)$ time by precomputing the set of edges in $R_i \cap R_j$ for each directly compatible pair (R_i, R_j) as well as the inverse relationship, namely the set of directly compatible pairs (R_i, R_j) whose intersection contains a given edge. We can also efficiently represent constraint subpath direct compatibility using a separate graph data structure G^c, where each node represents a constraint subpath and edges indicate direct compatibility. Then, maintaining the current "unioned" constraint subpaths along a given recursion path corresponds to maintaining a set of node-disjoint paths in this graph. These paths can be represented efficiently by tracking which subpaths are tails/heads of current paths and only permitting unions of a current head with a current tail (initially each node is in a separate path and is both a tail and a head). This can be checked and maintained in $O(1)$ time per directly compatible pair considered. When a directly compatible pair (R_i, R_j) containing e is found to union, we decrement the overdemand of all edges in $R_i \cap R_j$ by one; this takes $O(|E|)$ time. The initial work of building the constraint graph G^c can be done by checking each pair of subpath constraints for compatibility ($O(\ell^2|E|)$ time), and then finding the transitive reduction of this relation ($O(\ell^3)$ time using Aho's algorithm with standard matrix multiplication). Then for each edge $e \in G$, we can build a list of directly compatible constraint pairs that contain e in their intersection. ◀

3.2 A heuristic algorithm for MFDSC

In practice, we can run an MFD heuristic algorithm to determine a solution to the FD instance found via the reduction in the previous section. We use greedy-width, first proposed in [27], which greedily chooses the heaviest ("widest") paths in order to decompose the flow. As G' is a DAG, a greedy-width path can found in $O(|V| + |E| + \ell)$ time, by standard dynamic programming. In [27] it is shown that at most $|E| - |V| + 2$ greedy-width paths can be found, so the total time to find an FD solution is $O(|E|(|V| + |E| + \ell))$. Translating the FD solution back to the original graph (following Lemma 8) yields a path decomposition for the FDSC problem. However, in applications, we are often interested in finding solution to the MFDSC problem, i.e. finding a solution with the minimum number of paths. The introduction of bridge edges in the reduction described above may lead to more paths being required to decompose the reduced FD instance than the original FDSC instance. This is because we now must find paths through bridge edges, as well as in the original flow network. For this reason, we apply a *bridge reweighting* heuristic before decomposing the network in order to reduce the number of paths. For some arbitrary ordering of the bridge edges, we do the following:

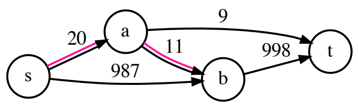
1. For each bridge edge, find the minimum flow $f_{\min}$ on the edges of its corresponding subpath constraint. Since the FDSC is feasible, $f_{\min} \geq 0$.
2. Subtract $f_{\min}$ from each of the subpath constraint edges, and add $f_{\min}$ to the bridge edge.

Since the bridge edge starts at the first node of the subpath constraint and ends at the last, flow conservation holds and the mapping of the bridge paths back to the original network again provides a solution to the FDSC instance. Figure 2 demonstrates the reduction to an FD instance and the bridge reweighting step on an example FDSC instance.

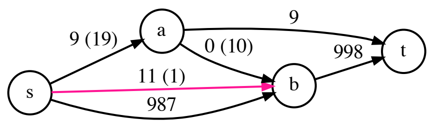
3.3 An FPT scheme for MFDSC

In this section, we describe an extension of Toboggan [13], an FPT algorithm for decomposing DAG flows, to also handle subpath constraints. Toboggan is able to find a k-path decomposition for a flow network $G = (V, E, f)$, if one exists, in $2^{O(k^2)} \cdot (|V| + \lambda))$ time, where λ is the logarithm of the largest flow value present. To solve MFD, Toboggan just tests increasing k values until a solution is found. We briefly describe Toboggan's approach and then discuss how to modify the algorithm so that it can also check if an FD solution also satisfies subpath constraints.

Toboggan considers the vertices of G in topological order and computes a table T_i for each vertex v_i using dynamic programming. Table entries are of the form (g, L), where g indicates how paths from the previous table T_{i-1} are extended, and L is a linear system



(a) An input FDSC instance with one subpath constraint, modified (for compactness) from the test dataset.



(b) The resulting FD instance that is solved using greedy-width. The pink edge is a *bridge edge* for the corresponding subpath constraint. Weights in parentheses are the weights before bridge reweighting.

Figure 2 Demonstration of reduction and bridge reweighthing procedure used in the heuristic MFDSC algorithm.

indicating how the weights of these paths are constrained to satisfy the flow requirements on all edges encountered so far. This linear system can be written as $\mathbf{A}\mathbf{w} = \mathbf{b}$, where $\mathbf{A}$ is a binary matrix of k columns representing whether each row's edge is covered by each column's path, $\mathbf{w}$ is the length k solution vector, and $\mathbf{b}$ is the flow on the row's edge. Because there are k weights and all coefficients are integers, each linear system can be reduced to k linearly independent rows. As noted in [13], testing an integer linear system L for feasibility and finding a solution can be done in $O(k^{2.5k+o(k)}|L|)$ time, where $|L|$ is shown to be $k^{O(1)}\lambda$.

When the final vertex in the order is reached, these linear systems indicate the path flow constraints on all edges in G, and so, if a particular system is feasible, the corresponding paths and weights provide an FD solution.

To modify Toboggan to also consider the subpath constraints, for the final table $T_{|V|}$, we will add a second linear system to simultaneously satisfy of the form $\mathbf{A}\mathbf{w} \geq \mathbf{b}$, where $\mathbf{A}$ is an $\ell \times k$ binary matrix and $\mathbf{b}^T = (d_1, \ldots, d_\ell)$. Here $A(i,j) \in \{0,1\}$, indicates whether path P_j contains R_i. We give an updated version of a lemma [13, Lemma 5] that bounds the number of distinct linear systems in the final table.

▶ **Lemma 18.** *The final table has at most $\frac{4^{k^2+\frac{k\ell}{2}}}{k!k^k}$ distinct linear systems.*

Proof. We follow the proof of [13, Lemma 5]. Since $\mathbf{A}$ is an $\ell \times k$ binary matrix, there are $2^{k\ell}$ possible systems of the second form. We must multiply this by the number of flow matching systems which was bounded ([13, Lemma 5]) by $\frac{4^{k^2}}{k!k^k}$. So, the total number of possible combined linear systems is $2^{k\ell}\frac{4^{k^2}}{k!k^k} = \frac{4^{k^2+\frac{k\ell}{2}}}{k!k^k}$. ◀

▶ **Theorem 19.** *Let $(G, \mathcal{R})$ be an FDSC instance with $|\mathcal{R}| = \ell$ and λ is the logarithm of the largest flow value in the input. Modifying the Toboggan algorithm as described provides an FPT algorithm for MFDSC with running time $2^{O(k^2)}|V| + 2^{O(k^2+k\ell)}(k+\ell)^{O(1)}\lambda$.*

Proof. Kloster et al. prove ([13, Lemma 4]), that in any table T_i, the number of distinct g values present is at most $\sqrt{k}(0.649k)^k$. This implies (following [13, Theorem 7]) that there are at most

$$\frac{4^{k^2+\frac{k\ell}{2}}}{k!k^k} \cdot \sqrt{k}(0.649k)^k = \sqrt{k}\frac{4^{k^2+\frac{k\ell}{2}}0.649^k}{k!}$$

final linear systems L to check for integer solutions. The encoding size of a linear system L is now bounded by $(k+\ell)^{O(1)}\lambda$, where λ is the logarithm of the largest flow value in the input. Checking feasibility and finding a solution for L can now be done in $O(k^{2.5k+o(k)}(k+\ell)^{O(1)}\lambda)$

time, so the total time needed to check all such linear systems is at most

$$\sqrt{k}\frac{4^{k^2+\frac{k\ell}{2}}0.649^k}{k!} \cdot O(k^{2.5k+o(k)}(k+\ell)^{O(1)}\lambda) \leq O(4^{k(k+\frac{\ell}{2})}k^{1.5k}1.765^k k^{o(k)}(k+\ell)^{O(1)}\lambda),$$

using the fact that $\frac{k^k}{k!} \leq e^k$. The total running time of the algorithm becomes $2^{O(k^2)}|V| + 2^{O(k^2+k\ell)}(k+\ell)^{O(1)}\lambda$. ◀

4 Experiments

The algorithms described in Section 3 were implemented in Python[1] in a package called *Coaster*.[2] We refer to the algorithm of Section 3.2 as *heuristic MFDSC* and Section 3.3 as *FPT MFDSC*. Experiments were performed on a high performance research cluster, where each run was executed on a single Intel Xeon Ivy Bridge E (3.4 Ghz) or similar CPU. For all runs, a memory limit of 20 Gb was set and instances were timed out if they ran longer than 30 seconds. For fairness of comparison, we report only on graph instances that ran to completion for all algorithm and parameter combinations, unless otherwise mentioned.

Datasets. As in previous studies on flow decomposition methods for RNA-Seq assembly [13, 22, 28], we use a simulated RNA-Seq dataset from [22] where each instance is a flow network generated by simulating RNA transcripts and their abundances with Flux-Simulator [10]. The original dataset includes human, mouse, and zebrafish genes, but we restrict our attention to instances in the human dataset, which contains 100 independently generated transcriptomes. As in [13, 28], we use only instances with at least two ground truth paths (since a single ground truth path is trivial to decompose). We also restrict the dataset to instances with 10 ground truth paths or fewer, yielding a total of 528,544 instances. Because the transcripts and abundances are known, we have ground truth paths and weights for each splice graph instance. We measure accuracy as the proportion of instances for which the algorithm returns the ground truth set of paths and weights exactly.

Simulating subpath constraints. In order to simulate subpath constraints, we take subpaths of the ground truth paths according to two parameters: the number of subpaths ℓ, and a fixed length for all subpaths $|R|$. As noted in [13, Lemma 8], we can simplify the graph by bypassing any vertex with out-degree or in-degree equal one. We set $|R|$ as the length of subpaths in this contracted graph. To generate subpath constraints that are consistent across experiments, we fix an arbitrary ordering for the ground truth paths for each instance, and take the first $|R|$ edges of the first ℓ (contracted) paths as the subpaths. We note that the method of generating subpath constraints described here does not yield any overdemanded edges.

Accuracy results. To study the effect of the subpath constraints on the accuracy of the RNA-Seq assembly, we vary ℓ and $|R|$ independently, letting $\ell \in [0, 4]$ and $|R| \in \{3, 4\}$. Because instances become more difficult to solve correctly as the number of ground truth paths increases, we separate results by the number of ground truth paths, which we denote by k. Accuracy results for both algorithms are reported in Table 1. For each k value, we

[1] Based on the codebase for Toboggan [12].
[2] Available at `https://github.com/msu-alglab/coaster`.

also report the percentage of instances that completed for all parameter combinations tested. As already shown by Kloster et al. [13], the MFD solutions found by Coaster for $\ell = 0$ (for them, Toboggan) do correspond to the the ground truth paths and weights most of the time. However, for larger k values, we can see that FPT MFD solutions (without subpath constraints) do not necessarily recover the correct set of paths and weights. For $k = 7$, for example, only 81% of the optimal decompositions produced by Coaster are the ground truth decomposition that we are seeking. Similarly, we see that FPT MFDSC solutions tend to be correct, with accuracy decreasing as k increased. However, FPT MFDSC has higher accuracy for all parameter combinations than FPT MFD at the same k value. For $k = 7$, when we add four subpath constraints of length four each, the ground truth decomposition is found 91% of the time, a 13% increase over FPT MFD.

When $\ell = 0$, our heuristic MFDSC algorithm is equivalent to the often-used greedy-width heuristic for MFD; our results show that adding subpath constraints to greedy-width increases its accuracy considerably for larger k values, for example, by 30% when $k = 7$. The increased accuracy of heuristic MFDSC is also good news for the use of MFDSC in practical methods, since heuristic MFD methods are already commonly used in RNA-Seq tools. In fact, the inclusion of many long subpath constraints makes heuristic MFDSC more accurate than FPT MFD for k values up 5, which account for 95.6% of the full dataset studied (all $k \geq 2$).

Part of the success of the heuristic MFDSC can be attributed to the fact that it finds optimal solutions in most cases. Without subpath constraints, heuristic MFDSC (i.e. greedy-width MFD) finds an optimal solution in 98.0% and the ground truth solution in 95.3% of instances in our dataset. With two subpath constraints of length 4, that increases to 99.0% and 98.1%, respectively. (For $\ell > 2$, small k values are excluded, so results are not comparable with $\ell = 0$ experiments.)

With and without subpath constraints, the vast majority of incorrectly predicted path decompositions are due to the algorithm returning an optimal decomposition of the same size as the ground truth one, but different from it, rather than a too-small optimal decomposition. As found in [13], in nearly all instances, the ground truth path decomposition is also an optimal decomposition. (They find that 0.043% of instances of all ground truth k that ran to completion in 50 seconds had non-optimal ground truth decompositions; we find that 0.100% of instances that completed in 30 seconds for all parameter combinations and had ground truth k less than 9 had non-optimal ground truth decompositions.) However, most instances are solved correctly, so it could be the case that the few instances that are not solved correctly are those that had non-optimal ground truths. This tends not to be the case. Overall, only 0.027% of instances for which the FPT for MFD yields incorrect solutions have non-optimal ground truth path decompositions. This is dominated by the $k = 2$ instances, however, for which no instance had a non-optimal ground truth; for $k = 3$ through $k = 8$, between 0.1% and 0.3% of instances that were predicted incorrectly had non-optimal ground truths. With many and longer subpath constraints ($|R| = 4$ and $\ell = 4$), it is still only a very small number – 0.052% – of incorrect solutions that have non-optimal ground truth path decompositions. Thus, this implies that the addition of subpath constraints restricts the solution space, allowing the algorithm to return the correct one more frequently and explaining the increase in accuracy when they are included.

Effect of the bridge reweighting. To confirm the effectiveness of the bridge reweighting heuristic for MFDSC, as opposed to simply using a path decomposition found by the method of Lemma 8, we measured the accuracy of the FDSC algorithm without bridge reweighthing on the same dataset studied above. In that case, the addition of subpath constraints in our

Table 1 Accuracy results using heuristic MFDSC (odd rows) and FPT MFDSC (even rows). For $k = 2$ through $k = 8$, we only report on instances that completed for every parameter combination; the "pc" column reports the percentage of instances that completed for all runs for each k value. For $k = 9$ and $k = 10$, less than 40% and 1% respectively of instances completed for the FPT runs, so we include them only for heuristic MFDSC, which ran to completion for all instances. The italicized values are the ones reported in [13, Figure 3], with some slight differences due to the fact that we restrict to the human dataset (they studied two additional datasets) and timeout differences.

				$\|R\| = 3$				$\|R\| = 4$			
k	n	pc	$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
2	291734	100%	0.992	0.999	0.999			1.000	1.000		
			0.991	0.999	0.999			1.000	1.000		
3	130867	100%	0.961	0.977	0.983	0.986		0.985	0.993	0.994	
			0.969	0.983	0.990	0.994		0.986	0.996	0.998	
4	58167	100%	0.901	0.926	0.941	0.948	0.958	0.942	0.964	0.974	0.979
			0.934	0.952	0.964	0.974	0.983	0.958	0.976	0.987	0.995
5	25933	100%	0.822	0.853	0.873	0.887	0.9	0.876	0.911	0.93	0.944
			0.892	0.913	0.928	0.940	0.953	0.922	0.944	0.962	0.976
6	11774	99.6%	0.727	0.763	0.784	0.805	0.816	0.787	0.831	0.862	0.883
			0.849	0.870	0.885	0.898	0.911	0.881	0.906	0.928	0.944
7	5095	94.6%	0.617	0.659	0.692	0.706	0.729	0.681	0.738	0.775	0.802
			0.810	0.835	0.855	0.871	0.883	0.845	0.872	0.894	0.912
8	2109	83.7%	0.495	0.523	0.558	0.589	0.611	0.545	0.607	0.664	0.702
			0.787	0.808	0.822	0.833	0.845	0.819	0.840	0.863	0.884
9	1323	100%	0.388	0.423	0.452	0.485	0.512	0.447	0.497	0.555	0.608
10	699	100%	0.310	0.333	0.349	0.367	0.393	0.351	0.383	0.418	0.454

experiments reduces the accuracy of the path decompositions returned, as shown in Table 2. Bridge reweighting allows the maximum flow that can cover a subpath constraint to do so, without introducing extra weight-one paths.

Performance results. We measured runtime and peak memory use of the implementation for both algorithms using all instances $k = 2$ through $k = 10$. For heuristic MFDSC on all instances, even those that timed out during FPT runs, the average, minimum, and maximum runtimes in seconds were 0.0059, 0.00096, and 0.977. For FPT instances that completed, they were 0.076, 0.001, and 30 (the timeout limit set for the experiments – instances would take longer if allowed). As mentioned in the Table 1 caption, for FPT MFDSC, less than half and almost no instances with $k = 9$ and $k = 10$ completed in 30 seconds respectively,

Table 2 Accuracy values for FD heuristic without bridge reweighthing. If all bridges are kept at weight one, subpath constraints reduce the accuracy of the path decomposition, though less if they are longer.

	$\|R\| = 3$				$\|R\| = 4$			
$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
0.980	0.940	0.816	0.588	0.348	0.958	0.919	0.798	0.637

indicating as expected that there is a steep increase in runtime as k gets large. However, for the heuristic MFDSC, no instance took longer than one second, even those with $k = 10$, showing that the heuristic MFDSC algorithm can be practical in applications. We measured the peak memory usage for batches of 2000 instances (batched according to the original order of instances in the dataset). Thus, the memory use for all instances is included, including those that timed out, and minimum and average measurements are over the batches, not individual instances. For heuristic MFDSC, the average, minimum, and maximum peak memory usage in MB was 52, 36, and 60. For FPT MFDSC, they were 77, 51, and 811.

5 Discussion

In this work, we initiate the formal study of the MFDSC problem, which is used as a model in applications such as RNA sequencing and viral quasispecies assembly. We give both a heuristic algorithm, based on a novel reduction to flow decomposition, and an FPT algorithm, which extends the FPT MFD algorithm of Kloster et al. [13]. Through experiments on a previously-studied simulated transcriptomics dataset, we verify the base assumption underlying the use of MFDSC in practical RNA-Seq tools: that the minimum-size path decomposition should correspond to the ground truth set of paths and weights. Additionally, we show that the use of subpath constraints increases accuracy when compared to MFD without subpath constraints. We also find that our heuristic algorithm is practical, completing in less than 1 second for all instances studied, and achieves accuracy levels near those of FPT MFDSC. This is an encouraging result, because while RNA sequencing data tends toward very small ground truth path sets, other multiassembly problems such as viral quasispecies assembly may not – for example, some benchmarking datasets of [3] contain 10 and 15 strains, meaning that MFDSC (or even MFD without subpath constraints) would be intractable without a heuristic.

The research presented here suggests a number of future directions. One is characterizing the complexity of determining whether an FDSC instance is feasible. The heuristic MFDSC algorithm we give begins with this step, but takes time exponential in the total overdemand and the number of subpath constraints if any edge is overdemanded. Another is to develop MFDSC algorithms for graphs containing cycles. Though splice graphs for RNA assembly are usually DAGs, graphs for de novo assembly of viral or other genomes would likely contain cycles due to repeated sequences. Finally, we find that as the size of ground truth gets large, accuracy decreases because there are multiple optimal solutions, even with the maximum length and number of subpath constraints that we tested. To increase accuracy, either more subpath constraints are needed (which may be possible, depending on the domain), or additional optimality criteria could be used.

References

1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

2 Jasmijn A. Baaijens, Bastiaan Van der Roest, Johannes Köster, Leen Stougie, and Alexander Schönhuth. Full-length de novo viral quasispecies assembly through variation graph construction. *Bioinform.*, 35(24):5086–5094, 2019. doi:10.1093/bioinformatics/btz443.

3 Jasmijn A Baaijens, Amal Zine El Aabidine, Eric Rivals, and Alexander Schönhuth. De novo assembly of viral quasispecies using overlap graphs. *Genome research*, 27(5):835–848, 2017.

4 Jasmijn A Baaijens, Leen Stougie, and Alexander Schönhuth. Strain-aware assembly of genomes from mixed samples using flow variation graphs. In *International Conference on Research in Computational Molecular Biology*, pages 221–222. Springer, 2020.

5 Georg Baier, Ekkehard Köhler, and Martin Skutella. On the k-splittable flow problem. In *European Symposium on Algorithms*, pages 101–113. Springer, 2002.

6 Georg Baier, Ekkehard Köhler, and Martin Skutella. The k-splittable flow problem. *Algorithmica*, 42(3-4):231–248, 2005.

7 Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs Theory, Algorithms and Applications*. Springer-Verlag, Berlin, 1st edition, 2000.

8 Ergude Bao, Tao Jiang, and Thomas Girke. Branch: boosting RNA-Seq assemblies with partial or related genomic sequences. *Bioinformatics*, 29(10):1250–1259, 2013.

9 Elsa Bernard, Laurent Jacob, Julien Mairal, and Jean-Philippe Vert. Efficient RNA isoform identification and quantification from RNA-Seq data with network flows. *Bioinformatics*, 30(17):2447–2455, 2014. doi:10.1093/bioinformatics/btu317.

10 Thasso Griebel, Benedikt Zacher, Paolo Ribeca, Emanuele Raineri, Vincent Lacroix, Roderic Guigó, and Michael Sammeth. Modelling and simulating generic RNA-Seq experiments with the flux simulator. *Nucleic acids research*, 40(20):10073–10083, 2012.

11 Tzvika Hartman, Avinatan Hassidim, Haim Kaplan, Danny Raz, and Michal Segalov. How to split a flow? In *2012 Proceedings IEEE INFOCOM*, pages 828–836. IEEE, 2012.

12 Kyle Kloster, Philipp Kuinke, Michael P O'Brien, Felix Reidl, Fernando Sánchez Villaamil, Blair D Sullivan, and Andrew van der Poel. Toboggan: Version 1.0, June 2017. doi:10.5281/zenodo.821634.

13 Kyle Kloster, Philipp Kuinke, Michael P O'Brien, Felix Reidl, Fernando Sánchez Villaamil, Blair D Sullivan, and Andrew van der Poel. A practical FPT algorithm for flow decomposition and transcript assembly. In *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 75–86. SIAM, 2018.

14 Anna Kuosmanen, Tuukka Norri, and Veli Mäkinen. Evaluating approaches to find exon chains based on long reads. *Briefings in bioinformatics*, 19(3):404–414, 2018.

15 Anna Kuosmanen, Ahmed Sobih, Romeo Rizzi, Veli Mäkinen, and Alexandru I. Tomescu. On using longer RNA-Seq reads to improve transcript prediction accuracy. In James P. Gilbert, Haim Azhari, Hesham H. Ali, Carla Quintão, Jan Sliwa, Carolina Ruiz, Ana L. N. Fred, and Hugo Gamboa, editors, *Proceedings of the 9th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2016) - Volume 3: BIOINFORMATICS, Rome, Italy, February 21-23, 2016*, pages 272–277. SciTePress, 2016. doi:10.5220/0005819702720277.

16 Wei Li, Jianxing Feng, and Tao Jiang. Isolasso: A LASSO regression approach to RNA-Seq based transcriptome assembly. *Journal of Computational Biology*, 18(11):1693–1707, 2011. doi:10.1089/cmb.2011.0171.

17 Brendan Mumey, Samareh Shahmohammadi, Kathryn McManus, and Sean Yaw. Parity balancing path flow decomposition and routing. In *2015 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2015.

18 Mihaela Pertea, Geo M Pertea, Corina M Antonescu, Tsung-Cheng Chang, Joshua T Mendell, and Steven L Salzberg. Stringtie enables improved reconstruction of a transcriptome from RNA-Seq reads. *Nature Biotechnology*, 33(3):290–295, 2015.

19 Krzysztof Pieńkosz and Kamil Kołtyś. Integral flow decomposition with minimum longest path length. *European Journal of Operational Research*, 247(2):414–420, 2015.

20 Romeo Rizzi, Alexandru I. Tomescu, and Veli Mäkinen. On the complexity of minimum path cover with subpath constraints for multi-assembly. *BMC Bioinform.*, 15(S-9):S5, 2014. doi:10.1186/1471-2105-15-S9-S5.

21 Mingfu Shao and Carl Kingsford. Accurate assembly of transcripts through phase-preserving graph decomposition. *Nature biotechnology*, 35(12):1167–1169, 2017.

22 Mingfu Shao and Carl Kingsford. Theory and a heuristic for the minimum path flow decomposition problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(2):658–670, 2017.

23 Tamara Steijger, Josep F Abril, Par G Engstrom, Felix Kokocinski, The RGASP Consortium, Tim J Hubbard, Roderic Guigo, Jennifer Harrow, and Paul Bertone. Assessment of transcript

reconstruction methods for RNA-Seq. *Nat Meth*, 10(12):1177–1184, December 2013. doi:10.1038/nmeth.2714.

24 Vorapong Suppakitpaisarn. An approximation algorithm for multiroute flow decomposition. *Electronic Notes in Discrete Mathematics*, 52:367–374, 2016. INOC 2015 – 7th International Network Optimization Conference. doi:10.1016/j.endm.2016.03.048.

25 Alexandru I. Tomescu, Anna Kuosmanen, Romeo Rizzi, and Veli Mäkinen. A novel min-cost flow method for estimating transcript expression with RNA-Seq. *BMC Bioinformatics*, 14(S-5):S15, 2013. Proceedings paper from RECOMB-seq: Third Annual RECOMB Satellite Workshop on Massively Parallel Sequencing Beijing, China. 11-12 April 2013. doi:10.1186/1471-2105-14-S5-S15.

26 Cole Trapnell, B.A. Williams, G. Pertea, Ali Mortazavi, G. Kwan, M.J. van Baren, S.L. Salzberg, B.J. Wold, and L. Pachter. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature Biotechnology*, 28:511–515, 2010.

27 Benedicte Vatinlen, Fabrice Chauvet, Philippe Chrétienne, and Philippe Mahey. Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. *European Journal of Operational Research*, 185(3):1390–1401, 2008.

28 Lucia Williams, Gill Reynolds, and Brendan Mumey. RNA transcript assembly using inexact flows. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1907–1914, 2019.

29 Ting Yu, Zengchao Mu, Zhaoyuan Fang, Xiaoping Liu, Xin Gao, and Juntao Liu. Transborrow: genome-guided transcriptome assembly by borrowing assemblies from different assemblers. *Genome Research*, 30(8):1181–1190, 2020. doi:10.1101/gr.257766.119.

30 Osvaldo Zagordi, Arnab Bhattacharya, Nicholas Eriksson, and Niko Beerenwinkel. ShoRAH: estimating the genetic diversity of a mixed sample from next-generation sequencing data. *BMC Bioinformatics*, 12(1):119+, 2011.