Flow Decomposition with Subpath Constraints

Lucia Williams, Alexandru I. Tomescu, Brendan Mumey

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.

Index Terms—Flow decomposition, subpath constraints, RNA-Seq.

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 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 subpaths in the 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)*.

In a version of this work presented at WABI 2021, we left open the question of whether FDSC (not necessarily minimal) can be solved in polynomial time. In this updated work, we give a polynomial time algorithm for FDSC, and present additional discussion on the hardness of two FDSC variants.

1.1 Biological setting

Algorithms that solve variations of the flow decomposition problem are at the heart of most RNA transcript assembly software, including IsoLasso [1], Traph [2], FlipFlop [3], Scallop [4] and StringTie [5]. More recently, flow decomposition methods were used for another multi-assembly problem, namely strain-aware genome assembly, with applications to viral quasispecies assembly [6], [7]. 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 [2]. Another approach [8] 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. [9] 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

- *Lucia Williams and Brendan Mumey are with the School of Computing, Montana State University, Bozeman, MT, USA. E-mail: brendan.mumey@montana.edu*
- *Alexandru Tomescu is with the Department of Computer Science, University of Helsinki, Finland.*

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 [10], [5]. Adding subpath constraints to the flow decomposition problem may further restrict the solution space, thus improving 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* [5] 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.

1.2 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 [11], and thus various practical approaches to it exist: approximation algorithms [12], [13], [14], [15], [16], [17], FPT algorithms [9], greedy algorithms [11], [18]. 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 Scallop [4] and StringTie [5]. 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 [6]. Recently, the method TransBorrow [19] 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 [20], and early virus quasispecies assembly methods such as ShoRAH [21]. The MPC problem has been extended to include subpath constraints as well [22], [23], [24], [25], 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 [26], [22], and thus the minimum FDSC problem is a strict generalization of the MPC problem with subpath constraints.

1.3 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 18), 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. Based on this reduction, in Section 3.2 we also propose a "bridge-reweighting" heuristic algorithm to solve the minimum FDSC problem. Additionally, we give an FPT algorithm for the minimum FDSC problem (Theorem 20), extending the one of Kloster et al. [9]. Finally, in Section 4, to add to the complexity picture around the FDSC problem, we show that two application-oriented FD problems related to FDSC are NP-hard in the strong sense, even without requiring a solution with a minimum number of paths.

We implement both algorithms for FDSC, and perform a proof-of-concept study of their usefulness in RNA assembly. We experiment on a dataset developed by Shao et al. [18] to study their heuristic for the minimum FD problem. The same dataset was then used by Kloster et al. [9], 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.

Our FDSC algorithm runs in polynomial time (i.e. always finds a solution to the FDSC problem, not necessarily minimum). Though we desire a minimum 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.

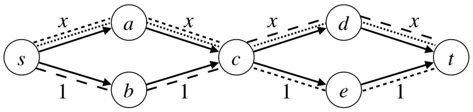


Fig. 1. An example FDSC flow network with the flow values of the edges being 1 or x; the dashed paths indicate the subpath constraints. If $x = 1$, then the instance is infeasible. If $x = 2$, then the instance is feasible and requires three paths to decompose (whereas the associated FD instance without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

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 $v \in V \setminus \{s, t\}$,

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

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 a 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 $e \in E$,

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

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 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

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

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

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 FDSC 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

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

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. We address this case in Section 3.1.1 and the case with overdemanded edges in Section 3.1.2.

3.1.1 Instances without overdemanded edges

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. □

Figure 3 shows an example of the reduction of an FDSC instance to a FD instance with a bridge edge.

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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2022.3147697, IEEE/ACM Transactions on Computational Biology and Bioinformatics

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 8.1. 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.

3.1.2 Resolving overdemanded edges

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. We now show how to obtain $(G, \mathcal{R}^*)$, if it exists.

Lemma 9. 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 is 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 $(G, \mathcal{R})$. □

Inspired by the above lemma, we consider pairs of subpath constraints that may be satisfied by the same path in a decomposition.

Definition 10 (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 11 (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 12. 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 9, 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 12.1. 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.

Lemma 12 suggests that we can determine feasibility by finding combinations of subpath constraints that are satisfied by the same paths. We can then think of merging these subpath constraints together to form an equivalent instance without overdemanded edges. One way to determine feasibility, then, would be to consider every possible way of merging subpaths; however, there are an exponential number of such possibilities.

To do this it polynomial time, we define a new graph built from the subpath constraints, and show that certain path coverings in this graph correspond to valid ways to merge the subpath constraints. If no such path cover exists, then the instance is infeasible.

We first define the new graph.

Definition 13 (Constraint graph)**.** Let $(G, \mathcal{R})$ be an FDSC instance. We define its *constraint graph* G^c as the graph where the vertices are constraints in $\mathcal{R}$ and an edge (R_i, R_j) indicates that R_i and R_j are directly compatible.

An example constraint graph for an FDSC instance is shown in Fig. 2.

Let L be the total length of all subpath constraints and recall that ℓ denotes the number of subpath constraints. We can construct the constraint graph G^c in $O(L + \ell^3)$ time by first using Gusfield's algorithm for all pairs suffix-prefix overlaps in $O(L + \ell^2)$ time [27] and then finding the transitive reduction of this relation in $O(\ell^3)$ time using Aho's algorithm with standard matrix multiplication [28]. (Note that the original all pairs suffix-prefix overlap relation is acyclic since no subpath constraint is properly contained inside another, and G is a DAG.)

Remark 14. G^c is a DAG.

Definition 15 (Edge-induced subgraph)**.** Let e be an edge in G. The *edge-induced subgraph* $G^c(e)$ is the subgraph of G^c consisting of all vertices R_i in G^c (and induced edges) such that $e \in R_i$.

We now show that a certain path cover of the constraint graph corresponds to a valid way to merge subpath constraints.

Lemma 16. Let $(G, \mathcal{R})$ be an FDSC instance and let G^c be its constraint graph. $(G, \mathcal{R})$ is feasible if and only if there is a vertex-disjoint path cover $\mathcal{P}^c$ of G^c such that, for every edge e in G, at most $f(e)$ paths in $\mathcal{P}^c$ visit $G^c(e)$.

Proof. $(\rightarrow)$ Assume $(G, \mathcal{R})$ is feasible. By Lemma 12 and Corollary 12.1, it is possible to merge subpath constraints until no edge is overdemanded. The resulting constraint subpaths are each formed from the union of original constraint subpaths in G that were directly compatible, so each resulting subpath corresponds to a path in G^c. Since each original subpath belongs to exactly one of the final subpaths, all such paths provide a vertex-disjoint path cover $\mathcal{P}^c$ of G^c. Let $e \in G$. Since e is not overdemanded, the number of paths from $\mathcal{P}^c$ that visit $G^c(e)$ is at most $f(e)$.

$(\leftarrow)$ Let $\mathcal{P}^c$ be a vertex-disjoint path cover for G^c such that, for every edge e in G, at most $f(e)$ paths in $\mathcal{P}^c$ visit $G^c(e)$. Create a new FDSC instance $(\mathcal{G}, \mathcal{R}')$ as follows: For each path in $\mathcal{P}^c$, add a single subpath constraint to $\mathcal{R}'$ that corresponds to union of the subpath constraints in the path. In $(\mathcal{G}, \mathcal{R}')$ no edge is overdemanded, since no edge in G had more associated paths in P^c than its flow. Thus $(\mathcal{G}, \mathcal{R}')$ is feasible and has a solution $(\mathcal{P}, w)$. Since any path covering a merged subpath constraint must cover the original unmerged subpath constraints it contains, $(\mathcal{P}, w)$ also provides a solution to $(G, \mathcal{R})$. □

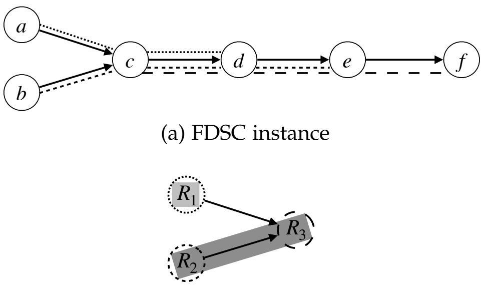
There is a simple greedy strategy (Algorithm 1 and Fig. 2) to find a vertex-disjoint path cover $\mathcal{P}^c$ of G^c that minimizes the number of paths intersecting $G^c(e)$ for all edges e in G.

Algorithm 1 Greedy algorithm to find a vertex-disjoint path cover for G^c such that for all edges e in G, the minimum possible number of paths in the cover visit $G^c(e)$.

```
function PathCover($(G, \mathcal{R})$, $G^c$)
    $V \leftarrow$ topological sorting $R_1, \ldots, R_\ell$ of vertices of $G^c$
    $\mathcal{P}^c \leftarrow \emptyset$
    for $R_i \in V$ do
        $U \leftarrow$ all in-neighbors of $R_i$
        if $|U| > 0$ then
            $u_i \leftarrow$ the $u \in U$ with greatest number of edges in suffix-prefix overlap with $R_i$
            Extend the path ending at $u_i$ to end at $R_i$
        else
            Add new path starting and ending at $R_i$ to $\mathcal{P}^c$
        end if
    end for
    return $\mathcal{P}^c$
end function
```

Lemma 17. In $O(\ell^2)$ time, Algorithm 1 finds a vertex-disjoint path cover for G^c such that for every edge e in G, the minimum possible number of paths in the cover visit $G^c(e)$.

Proof. Consider any vertex R in G^c, with incoming edges from $R_1, \ldots, R_a$. Clearly, at most one such edge (R_i, R) can belong to any vertex-disjoint path cover. Observe that for any edge e in G, whether (R_i, R) belongs to a path in the cover only affects the number of cover paths visiting $G^c(e)$ provided both R_i and R belong to $G^c(e)$; in other words, e belongs to both R_i and R. For each incoming edge (R_i, R) in G^c, consider the set of $e \in G$ for which e belongs to both R_i and R; these edges form a path in G that is a prefix of R. Thus, choosing (R_i, R) will benefit — i.e., not increase — the visitation counts to exactly those edges in this prefix of R. It follows that simply choosing the (R_i, R) that has the longest suffix-prefix overlap will minimize all visitation counts. By considering the vertices in topological order, we can extend paths in $O(1)$ time.



(a) FDSC instance

(b) Constraint graph

Fig. 2. Illustration of the greedy choice for extending paths used in Algorithm 1. When processing node R_3 in the constraint graph in Figure 2b, the path from R_2 is extended because the overlap between R_2's subpath constraint and R_3's subpath constraint is greater than R_1's and R_3's.

Since G^c has ℓ vertices, we can perform the topological sort in $O(\ell^2)$ time. Each edge is examined once during the algorithm, and we assume that suffix-prefix overlaps have been pre-computed during the construction of G^c, so the total running time of this step is $O(\ell^2)$.

□

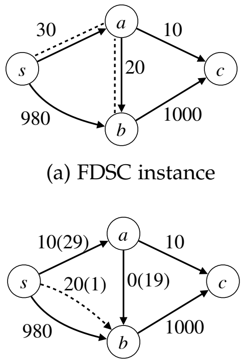
Because we can find an optimal path cover in polynomial time, we can check the feasibility of an FDSC instance (and, if it is feasible, find a solution) in polynomial time.

Theorem 18. Let $(G, \mathcal{R})$ be an FDSC instance with $|\mathcal{R}| = \ell$, total length of all subpath constraints L, and at least one overdemanded edge. In $O(L + \ell^3)$ time, we can determine whether $(G, \mathcal{R})$ is feasible, and if so, reduce $(G, \mathcal{R})$ to an equivalent FD instance with at most ℓ additional edges.

Proof. As discussed above, G^c and the suffix-prefix overlaps can be found in $O(L+\ell^3)$ time. We can then use Algorithm 1 to find an optimal vertex-disjoint path cover P^c of G^c in $O(\ell^2)$ time. Because the path cover is optimal, Lemma 16 tells us that $(G, \mathcal{R})$ is feasible if and only if for every edge e of G, at most $f(e)$ paths in P^c visit $G^c(e)$; this can be checked in $O(L)$ time. If the instance is feasible, by Lemma 16, we can merge the subpath constraints in each path found by Algorithm 1, yielding an equivalent FDSC instance with no overdemanded edges. Then, we can use the method of Lemma 7 to transform that FDSC instance into an equivalent FD instance with at most ℓ additional edges. □

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 [11], which greedily chooses the heaviest ("widest") paths



(a) FDSC instance

(b) Reduced graph

Fig. 3. Demonstration of reduction and bridge reweighthing procedure used in the heuristic MFDSC algorithm. The resulting FD instance in Figure 3b is solved using greedy-width. The dashed edge is a *bridge edge* for the corresponding subpath constraint. Weights in parentheses are the weights before bridge reweighting.

in order to decompose the flow[1]. As G' is a DAG, a greedy-width path can found in $O(|V|+|E|+\ell)$ time, by standard dynamic programming. In [11] 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 $f_{\min}$ over the flow values on the edges of its corresponding subpath constraint in the original network. 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 3 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 [9], 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 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 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 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 [9], 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 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 [9, Lemma 5] that bounds the number of distinct linear systems in the final table.

Lemma 19. 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 [9, 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 ([9, 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 20. 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 ([9, 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 [9, 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

[1] Greedy-width is a common heuristic algorithm for MFD. Other heuristics such as Catfish [18] could also be substituted.

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

$$\begin{aligned}\sqrt{k}\frac{4^{k^2+\frac{k\ell}{2}}0.649^k}{k!} &\cdot O(k^{2.5k+o(k)}(k+\ell)^{O(1)}\lambda) \\ &\leq O(4^{k(k+\frac{\ell}{2})}k^{1.5k}1.765^k k^{o(k)}(k+\ell)^{O(1)}\lambda), \quad (5)\end{aligned}$$

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 Hardness of Related Flow Decomposition Problems

In this section we add to the computational complexity picture around the FDSC problem by studying two natural application-oriented variants of it. In contrast to the FDSC problem, where deciding if the instance is feasible can be done in polynomial time (recall Theorem 18), for both of these problems feasibility checking is NP-complete in the strong sense (i.e. even if the input flow values are bounded by a polynomial in the size of the input; in fact, the former one is NP-complete even when all flow values equal 1). Since their application setting is slightly different from the one of the FDSC problem (see our FDSC experiments in Section 5), in this paper we do not give algorithms for them, and leave that for future work.

First, we consider the version of the problem where the subpath constraints have associated *demands* that must be met by the flow assigned to a path that covers them. This problem could arise if we would like a subpath constraint to be covered by at least one flow path of higher weight, since a more heavily-weighted path may be less likely to be result of noisy data.

Definition 21 (Flow decomposition with subpath constraints and demands)**.** Let $G = (V, E, f)$ be a flow network. Let $\mathcal{R} = \{R_1, \ldots, R_\ell\}$ be a set of subpath constraints in G, and let $\mathcal{D} = \{d_1, \ldots, d_\ell\}$ be a set of *demands*, where each d_i is associated to subpath constraint R_i. We say that a flow decomposition $(\mathcal{P}, w)$ of G satisfies subpath constraints $\mathcal{R}$ and demands $\mathcal{D}$ if and only if

$$\forall R_i \in \mathcal{R}\ \exists P_j \in \mathcal{P} \text{ such that } R_i \in P_j \text{ and } d_i \leq w_j. \quad (6)$$

Problem 22 (FDSCD)**.** Given a flow network $G = (V, E, f)$ and subpaths constraints $\mathcal{R}$ and demands $\mathcal{D}$, the *flow decomposition with subpaths constraints and demands* problem is to determine if there exists, and if so, find a flow decomposition $(\mathcal{P}, w)$ satisfying (6).

We note that FDSC is a special case of FDSCD, where all demands are equal to one. The following proof is very similar to the NP-completeness proof of [11] for MFD.

Theorem 23. FDSCD is NP-complete in the strong sense.

Proof. Clearly, FDSCD belongs to NP. For NP-hardness, we reduce from the 3-PARTITION problem. The input of this problem consists of a set of positive integers $A = \{a_1, a_2, \ldots, a_{3q}\}$, where $\sum_{a\in A} a = qB$ and $B/4 < a < B/2$ for all $a \in A$. The question is whether there exists a partition of A into q disjoint subsets, each with exactly three elements summing up to B.

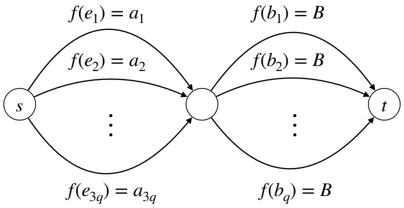


Fig. 4. Given an instance of 3-PARTITION $A = \{a_1, a_2, \ldots, a_{3q}\}$ with $\sum_{a\in A} a = qB$, we construct the flow network with edges $e_1, \ldots, e_{3q}$, $b_1, \ldots, b_q$. For all $i \in \{1, \ldots, 3q\}$, we set $f(e_i) = a_i$, and for all $j \in \{1, \ldots, q\}$ we set $f(b_j) = B$. For each e_i we add the single-edge subpath constraint $[e_i]$ with demand $d_i = a_i$.

Given an input for 3-PARTITION, we construct the flow network $G = (V, E, f)$ with subpath constraints $\mathcal{R}$ with demands $\mathcal{D}$ as in Figure 4. We claim that this is a YES instance for 3-PARTITION if and only if the reduction creates a YES-instance for FDSCD.

$(\rightarrow)$ Assume $\{\{a_{j_1}, a_{j_2}, a_{j_3} \mid j \in \{1, \ldots, q\}\}\}$ is a proper 3-partition of A. For each $j \in \{1, \ldots, q\}$, we build the three flow paths (e_{j_1}, b_j), (e_{j_2}, b_j), (e_{j_3}, b_j), with weights $a_{j_1}, a_{j_2}, a_{j_3}$, respectively. This is possible since $f(b_j) = B = \sum_{k=1}^{3} a_{j_k}$. As such, each subpath constraint $[e_i]$ of demand a_i is satisfied.

$(\leftarrow)$ Let $(\mathcal{P}, w)$ be a flow decomposition with subpath constraints $\mathcal{R}$ and demands $\mathcal{D}$ of G as indicated. Since the demand of each constraint $[e_i]$ is a_i, and $f(e_i) = a_i$ it follows that each edge e_i is used by exactly one flow path of weight a_i, and thus that $\mathcal{P}$ consists of exactly $3q$ paths $P_1, \ldots, P_{3q}$, with weights $a_1, a_2, \ldots, a_{3q}$, respectively. For each $j \in \{1, \ldots, q\}$, consider the flow paths passing thorough b_j. Since $B/4 < a < B/2$ holds for all $a \in A$, there are exactly 3 such paths, say $P_{j_1}, P_{j_2}, P_{j_3}$, each of weight $a_{j_1}, a_{j_2}, a_{j_3}$ respectively, passing through b_j. As such, $\{\{a_{j_1}, a_{j_2}, a_{j_3} \mid j \in \{1, \ldots, q\}\}\}$ is a proper 3-partition of A.

Finally, since 3-PARTITION is NP-complete in the strong sense [29], it follows that FDSCD is as well. □

Our second problem variant is motivated by paired-end reads, which naturally induce *pairs* of subpath constraints that must be covered by the same flow path (since e.g. they are sequenced from the same RNA transcript).

Definition 24 (Flow decomposition with paired subpath constraints)**.** Let $G = (V, E, f)$ be a flow network. *Paired subpaths constraints* are defined to be a set of pairs of simple paths $\mathcal{R} = \{(R_1, R_1'), \ldots, (R_\ell, R_\ell')\}$ in G. A flow decomposition $(\mathcal{P}, w)$ satisfies the paired subpaths constraints if and only if

$$\forall (R_i, R_i') \in \mathcal{R}\ \exists P_j \in \mathcal{P} \text{ such that } R_i \in P_j \text{ and } R_i' \in P_j. \quad (7)$$

Problem 25 (FDPSC)**.** Given a flow network $G = (V, E, f)$ and paired subpaths constraints $\mathcal{R}$, the *flow decomposition with paired subpaths constraints* problem is to determine if there exists, and if so, find a flow decomposition $(\mathcal{P}, w)$ satisfying (7).

Our NP-completeness proof below is similar to the NP-completeness proof of the minimum path cover problem with paired subpath constraints from [30]. Note that hardness holds even if all flow values equal 1.

Theorem 26. FDPSC is NP-complete even when all flows values are 1.

Proof. Clearly, FDPSC belongs to NP. For NP-hardness, we reduce from the problem of deciding whether the chromatic number of an undirected graph $G = (V, E)$ is 3. Given G, we construct the flow network $G^* = (V^*, E^*, f)$, with paired subpaths constraints $\mathcal{R}$, as in Figure 5. We claim that the chromatic number of G is 3 if and only if $(G^*, \mathcal{R})$ is a feasible FDPSC instance.

$(\rightarrow)$ Let V_1, V_2, V_3 be a partition of V such that no edge of G has endpoints in the same V_k. Then, we can construct a flow decomposition (P_1, P_2, P_3) of G^*, where each path has weight 1, as follows. In the vertex part of G^*, suppose $v_i \in V_k$; P_k follows the edge labeled v_i, and the other two paths follow the other two edges parallel to it, respectively. In the edge part of G^*, for each edge $e = (v_i, v_j)$, where $v_i \in V_{k_i}$ and $v_j \in V_{k_j}$, path P_{k_i} follows the edge labeled v_i' and path P_{k_j} follows the edge labeled v_j'. The remaining path follows the middle unlabeled edge. Clearly, all subpath constraint pairs are satisfied.

$(\leftarrow)$ Any flow decomposition of G^* has exactly 3 paths, each of weight 1. Let P_1, P_2, P_3 be a flow decomposition with paired subpaths constraints of $(G^*, \mathcal{R})$. We construct a partition V_1, V_2, V_3 of V: in the vertex part of G^*, an edge labeled v_i belongs exactly to one P_k, and we assign v_i to V_k. Assume for a contradiction that some edge $e = (v_i, v_j)$ of G has endpoints in the same V_k. Recall that for e, $\mathcal{R}$ contains a constraint pairing the edge labeled v_i with the one labeled v_i', and one constraint pairing the edge labeled v_j with the one labeled v_j'. Because of these constraints, since P_k passes through both v_i and v_j (in the vertex part of G^*), in the edge part of G^* corresponding to e we have that P_k passed through both the edge labeled v_i' and the one labeled v_j', which is a contradiction. □

5 Experiments

The algorithms described in Section 3 were implemented in Python 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. We denote the number of groundtruth paths for an instance by k, and set a CPU time limit of 30 seconds for smaller instances ($2 \leq k \leq 8$) and 1 hour for larger instances ($k = 9, 10$). For fairness of comparison, we report only on graph instances that ran to completion for all algorithm and parameter combinations, unless otherwise mentioned. Overall, we find that heuristic MFDSC completes for all test instances in under one second, and FPT MFDSC completes in under 30 seconds for all instances with $k \leq 5$, which includes the majority of instances. We give additional details and results in the following sections.

5.1 Datasets

As in previous studies on flow decomposition methods for RNA-Seq assembly [9], [18], [8], we use a simulated RNA-Seq dataset from [18] where each instance is a flow network generated by simulating RNA transcripts and their abundances with Flux-Simulator [32]. 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 [9], [8], 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.

5.2 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 [9, 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.

5.3 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 k. Accuracy results for both algorithms are reported in Table 1. For each k value, we also report the percentage of instances that completed for all parameter combinations tested. As already shown by Kloster et al. [9], 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

[2] Coaster is based on the codebase for Toboggan [31] and is available at github.com/msu-alglab/coaster.

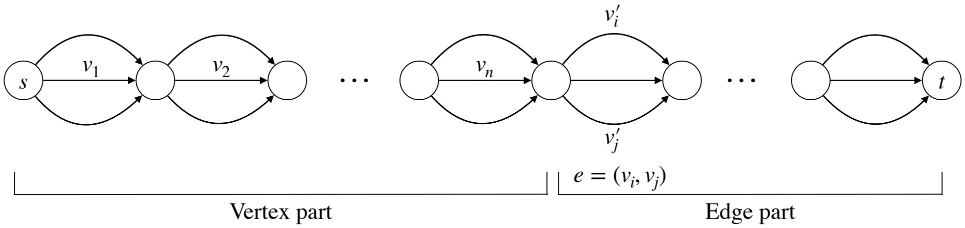


Fig. 5. Given an undirected graph $G = (V, E)$, with $V = \{v_1, v_2, \ldots, v_n\}$, we construct a flow network G^* as illustrated. For every vertex v_i we add three parallel edges, labelling one of them with v_i. For every edge $e = (v_i, v_j)$ of G, we add three parallel edges, one labeled v_i' and one labeled v_j'. For e we also add one constraint pairing the edge labeled v_i from the vertex part, with the edge labeled v_i', and one constraint pairing the edge labeled v_j from the vertex part with the edge labeled v_j'. The flow value of every edge is 1.

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 to 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 [9], 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.

5.4 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 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.

5.5 Performance results

We measured CPU runtime of the implementation for both algorithms using all instances for the given k range, even those that timed out for some experimental conditions. For heuristic MFDSC on $2 \leq k \leq 8$ instances, the average, minimum, and maximum runtimes were 0.0059s, 0.00096s, and 0.977s. For FPT instances, they were 0.076s, 0.001s, and 30s (the maximum time allowed). On $k = 9, 10$ instances, the average, minimum, and maximum runtimes were 0.018s, 0.004s, 0.155s for heuristic MFDSC and 289.3s, 0.932s, and 3600s (the maximum time allowed) for FPT FDSC.

Because of the optimizations made in Toboggan [9] on which our FPT MFDSC implementation is based, memory use is generally very limited even as k increases. We measured the peak memory use for large instances ($k = 9, 10$) with a timeout limit of 1 hour for all experimental combinations (ℓ and $|R|$ values) for the FPT MFDSC algorithm. All but four instances used under 100MB of memory in all experimental combinations; the average memory use over all instances and all experimental combinations, even those that timed out after an hour, was 47MB. For most instances, the memory use is dominated by loading the required Python packages (about 40MB).

6 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. [9]. Through experiments on a previously-studied simulated transcriptomics dataset, we verify the base assumption underlying the use of

TABLE 1
Accuracy results using the MFDSC with bridge-reweighting heuristic algorithm (labeled "H-br") and the MFDSC FPT algorithm (labeled "FPT"). For $k = 2$ through $k = 8$ we use a CPU time limit of 30 seconds; for $k = 9, 10$ we use 1 hour. We only report instances that finished in the time limit for all $\ell, |R|$, and for both algorithms for each k; the "pc" column reports the percentage of instances that completed for all runs for each k value. The italicized values agree with the ones reported in [9, 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.

					$\lvert R\rvert = 3$				$\lvert R\rvert = 4$			
k	n	pc	alg	$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
2	291734	100%	H-br	0.992	0.999	0.999			1.000	1.000		
			FPT	*0.991*	0.999	0.999			1.000	1.000		
3	130867	100%	H-br	0.961	0.977	0.983	0.986		0.985	0.993	0.994	
			FPT	*0.969*	0.983	0.990	0.994		0.986	0.996	0.998	
4	58167	100%	H-br	0.901	0.926	0.941	0.948	0.958	0.942	0.964	0.974	0.979
			FPT	*0.934*	0.952	0.964	0.974	0.983	0.958	0.976	0.987	0.995
5	25933	100%	H-br	0.822	0.853	0.873	0.887	0.900	0.876	0.911	0.930	0.944
			FPT	*0.892*	0.913	0.928	0.940	0.953	0.922	0.944	0.962	0.976
6	11774	99.6%	H-br	0.727	0.763	0.784	0.805	0.816	0.787	0.831	0.862	0.883
			FPT	*0.849*	0.870	0.885	0.898	0.911	0.881	0.906	0.928	0.944
7	5095	94.6%	H-br	0.617	0.659	0.692	0.706	0.729	0.681	0.738	0.775	0.802
			FPT	*0.810*	0.835	0.855	0.871	0.883	0.845	0.872	0.894	0.912
8	2109	83.7%	H-br	0.495	0.523	0.558	0.589	0.611	0.545	0.607	0.664	0.702
			FPT	*0.787*	0.808	0.822	0.833	0.845	0.819	0.840	0.863	0.884
9	1323	83.14%	H-br	0.455	0.495	0.527	0.565	0.592	0.522	0.582	0.643	0.698
			FPT	*0.714*	0.731	0.742	0.762	0.781	0.745	0.769	0.795	0.821
10	699	69.53%	H-br	0.420	0.442	0.459	0.484	0.508	0.465	0.506	0.541	0.578
			FPT	*0.726*	0.747	0.757	0.772	0.784	0.757	0.776	0.807	0.825

TABLE 2
Accuracy values for FD heuristic with ("H-br") and without ("H-b") bridge reweighthing, averaged over all instances with $k \leq 2 \leq 10$. If all bridges are kept at weight one, subpath constraints reduce the accuracy of the path decomposition, though less if they are longer.

		$\lvert R\rvert = 3$				$\lvert R\rvert = 4$			
	$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
H-br	0.951	0.965	0.970	0.942	0.899	0.971	0.979	0.963	0.930
H-b	0.951	0.926	0.809	0.582	0.379	0.945	0.913	0.793	0.633

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 [33] 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 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. Another is to explore additional methods to increase the accuracy of the decompositions found. Our experiments show that as the size of ground truth gets large, accuracy decreases because there are multiple optimal solutions to choose from, 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. For example, the two FDSC variants considered in Section 4 (together with yet to be discovered algorithms for them and possible problem generalizations) could guide the search for such optimality criteria.

References

[1] W. Li, J. Feng, and T. Jiang, "Isolasso: A LASSO regression approach to RNA-Seq based transcriptome assembly," *Journal of Computational Biology*, vol. 18, no. 11, pp. 1693–1707, 2011. [Online]. Available: http://dx.doi.org/10.1089/cmb.2011.0171

[2] A. I. Tomescu, A. Kuosmanen, R. Rizzi, and V. Mäkinen, "A novel min-cost flow method for estimating transcript expression with RNA-Seq," *BMC Bioinformatics*, vol. 14, no. S-5, p. S15, 2013, proceedings paper from RECOMB-seq: Third Annual RECOMB Satellite Workshop on Massively Parallel Sequencing Beijing, China. 11-12 April 2013. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-14-S5-S15

[3] E. Bernard, L. Jacob, J. Mairal, and J. Vert, "Efficient RNA isoform identification and quantification from RNA-Seq data with network flows," *Bioinformatics*, vol. 30, no. 17, pp. 2447–2455, 2014. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btu317

[4] M. Shao and C. Kingsford, "Accurate assembly of transcripts through phase-preserving graph decomposition," *Nature biotechnology*, vol. 35, no. 12, pp. 1167–1169, 2017.

[5] M. Pertea, G. M. Pertea, C. M. Antonescu, T.-C. Chang, J. T. Mendell, and S. L. Salzberg, "Stringtie enables improved reconstruction of a transcriptome from RNA-Seq reads," *Nature Biotechnology*, vol. 33, no. 3, pp. 290–295, 2015.

[6] J. A. Baaijens, L. Stougie, and A. Schönhuth, "Strain-aware assembly of genomes from mixed samples using flow variation graphs," in *International Conference on Research in Computational Molecular Biology*. Springer, 2020, pp. 221–222.

[7] J. A. Baaijens, B. V. der Roest, J. Köster, L. Stougie, and A. Schönhuth, "Full-length de novo viral quasispecies assembly through variation graph construction," *Bioinform.*, vol. 35, no. 24, pp. 5086–5094, 2019. [Online]. Available: https://doi.org/10.1093/bioinformatics/btz443

[8] L. Williams, G. Reynolds, and B. Mumey, "RNA transcript assembly using inexact flows," in *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2019, pp. 1907–1914.

[9] K. Kloster, P. Kuinke, M. P. O'Brien, F. Reidl, F. S. Villaamil, B. D. Sullivan, and A. 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)*. SIAM, 2018, pp. 75–86.

[10] T. Steijger, J. F. Abril, P. G. Engstrom, F. Kokocinski, T. R. Consortium, T. J. Hubbard, R. Guigo, J. Harrow, and P. Bertone, "Assessment of transcript reconstruction methods for RNA-Seq," *Nat Meth*, vol. 10, no. 12, pp. 1177–1184, 12 2013. [Online]. Available: http://dx.doi.org/10.1038/nmeth.2714

[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, no. 3, pp. 1390–1401, 2008.

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

[13] V. Suppakitpaisarn, "An approximation algorithm for multiroute flow decomposition," *Electronic Notes in Discrete Mathematics*, vol. 52, pp. 367 – 374, 2016, iNOC 2015 – 7th International Network Optimization Conference. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1571065316300531

[14] K. Pieńkosz and K. Kołtyś, "Integral flow decomposition with minimum longest path length," *European Journal of Operational Research*, vol. 247, no. 2, pp. 414–420, 2015.

[15] B. Mumey, S. Shahmohammadi, K. McManus, and S. Yaw, "Parity balancing path flow decomposition and routing," in *2015 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2015, pp. 1–6.

[16] G. Baier, E. Köhler, and M. Skutella, "On the k-splittable flow problem," in *European Symposium on Algorithms*. Springer, 2002, pp. 101–113.

[17] ——, "The k-splittable flow problem," *Algorithmica*, vol. 42, no. 3-4, pp. 231–248, 2005.

[18] 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, 2017.

[19] T. Yu, Z. Mu, Z. Fang, X. Liu, X. Gao, and J. Liu, "Transborrow: genome-guided transcriptome assembly by borrowing assemblies from different assemblers," *Genome Research*, vol. 30, no. 8, pp. 1181–1190, 2020. [Online]. Available: http://genome.cshlp.org/content/30/8/1181.abstract

[20] C. Trapnell, B. Williams, G. Pertea, A. Mortazavi, G. Kwan, M. van Baren, S. Salzberg, B. Wold, and L. Pachter, "Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation," *Nature Biotechnology*, vol. 28, pp. 511–515, 2010.

[21] O. Zagordi, A. Bhattacharya, N. Eriksson, and N. Beerenwinkel, "ShoRAH: estimating the genetic diversity of a mixed sample from next-generation sequencing data." *BMC Bioinformatics*, vol. 12, no. 1, pp. 119+, 2011.

[22] R. Rizzi, A. I. Tomescu, and V. Mäkinen, "On the complexity of minimum path cover with subpath constraints for multi-assembly," *BMC Bioinform.*, vol. 15, no. S-9, p. S5, 2014. [Online]. Available: https://doi.org/10.1186/1471-2105-15-S9-S5

[23] A. Kuosmanen, A. Sobih, R. Rizzi, V. Mäkinen, and A. I. Tomescu, "On using longer RNA-Seq reads to improve transcript prediction accuracy," in *Proceedings of the 9th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2016) - Volume 3: BIOINFORMATICS, Rome, Italy, February 21-23, 2016*, J. P. Gilbert, H. Azhari, H. H. Ali, C. Quintão, J. Sliwa, C. Ruiz, A. L. N. Fred, and H. Gamboa, Eds. SciTePress, 2016, pp. 272–277. [Online]. Available: https://doi.org/10.5220/0005819702720277

[24] E. Bao, T. Jiang, and T. Girke, "Branch: boosting RNA-Seq assemblies with partial or related genomic sequences." *Bioinformatics*, vol. 29, no. 10, pp. 1250–1259, 2013.

[25] A. Kuosmanen, T. Norri, and V. Mäkinen, "Evaluating approaches to find exon chains based on long reads," *Briefings in bioinformatics*, vol. 19, no. 3, pp. 404–414, 2018.

[26] J. Bang-Jensen and G. Z. Gutin, *Digraphs Theory, Algorithms and Applications*, 1st ed. Berlin: Springer-Verlag, 2000.

[27] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. New York, NY, USA: Cambridge University Press, 1997.

[28] A. V. Aho, M. R. Garey, and J. D. Ullman, "The transitive reduction of a directed graph," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 131–137, 1972.

[29] M. R. Garey and D. S. Johnson, "Complexity results for multiprocessor scheduling under resource constraints," *SIAM Journal on Computing*, vol. 4, no. 4, pp. 397–411, 1975.

[30] R. Rizzi, A. I. Tomescu, and V. Mäkinen, "On the complexity of minimum path cover with subpath constraints for multi-assembly," in *BMC Bioinformatics*, vol. 15, no. S9. Springer, 2014, p. S5.

[31] K. Kloster, P. Kuinke, M. P. O'Brien, F. Reidl, F. S. Villaamil, B. D. Sullivan, and A. van der Poel, "Toboggan: Version 1.0," Jun. 2017. [Online]. Available: http://dx.doi.org/10.5281/zenodo.821634

[32] T. Griebel, B. Zacher, P. Ribeca, E. Raineri, V. Lacroix, R. Guigó, and M. Sammeth, "Modelling and simulating generic RNA-Seq experiments with the flux simulator," *Nucleic acids research*, vol. 40, no. 20, pp. 10 073–10 083, 2012.

[33] J. A. Baaijens, A. Z. El Aabidine, E. Rivals, and A. Schönhuth, "De novo assembly of viral quasispecies using overlap graphs," *Genome research*, vol. 27, no. 5, pp. 835–848, 2017.

Brendan Mumey obtained his Ph.D. in Computer Science and Engineering at the University of Washington in 1997 and joined the faculty at Montana State University in 1998 where he currently serves as Professor of Computer Science. In the spring of 2020, he held a visiting Fulbright-Nokia Distinguished Chair position at the University of Helsinki. He served as the Editor of ACM SIGACT News, a quarterly periodical for the theoretical computer science community, from 2008 to 2015.

Alexandru I. Tomescu obtained his PhD in Computer Science in 2012 at the University of Udine, Italy. Between 2012 and 2020, he worked at the University of Helsinki in several postdoctoral positions on Algorithmic Bioinformatics, including Academy of Finland Postdoctoral Fellow (2014–2017), and Researcher (2019–). In 2020 he was appointed as Associate Professor at the University of Helsinki, where he currently leads the Graph Algorithms team, of the wider Algorithmic Bioinformatics research group. In 2019 he obtained the ERC Starting Grant.

Lucia Williams is pursuing a Ph.D. in Computer Science at Montana State University. She completed her B.S. at the University of Washington in 2014.