Safety in Multi-Assembly via Paths Appearing in All Path Covers of a DAG

Manuel Cáceres, Brendan Mumey, Edin Husić, Romeo Rizzi, Massimo Cairo, Kristoffer Sahlin, and Alexandru I. Tomescu

Abstract—A *multi-assembly problem* asks to reconstruct *multiple* genomic sequences from mixed reads sequenced from all of them. Standard formulations of such problems model a solution as a path cover in a directed acyclic graph, namely a set of paths that together cover all vertices of the graph. Since multi-assembly problems admit multiple solutions in practice, we consider an approach commonly used in standard genome assembly: output only partial solutions (*contigs*, or *safe paths*), that appear in all path cover solutions. We study *constrained* path covers, a restriction on the path cover solution that incorporate practical constraints arising in multi-assembly problems. We give efficient algorithms finding *all* maximal safe paths for constrained path covers. We compute the safe paths of splicing graphs constructed from transcript annotations of different species. Our algorithms run in less than 15 seconds per species and report *RNA contigs* that are over 99% precise and are up to 8 times longer than unitigs. Moreover, RNA contigs cover over 70% of the transcripts and their coding sequences in most cases. With their increased length to unitigs, high precision, and fast construction time, maximal safe paths can provide a better base set of sequences for transcript assembly programs.

Index Terms—Graph algorithms, network problems, analysis of algorithms and problem complexity, biology and genetics

1 Introduction

Many real-world problems require to reconstruct an unknown object from partial data observed from it. *Genome assembly* is a typical instance of such problem in Bioinformatics: given a set of high-throughput sequencing reads obtained from some genomic sequence, we need to reconstruct the sequence from which the reads originate. A major issue in such problems is that multiple solutions (reconstructions) can explain the observed data, making it difficult to distinguish the correct solution. As such, reporting one arbitrary solution may easily lead to an incorrect answer to the problem. An established way of coping with this issue is to report only partial solutions about which we are "confident" that they are correct. For example, state-of-the-art genome assemblers do not output entire chromosomes, but only *contigs*, namely genomic fragments that are promised to occur in the original genome.

An algorithmic way of formalizing such "reliable" partial solutions is through the notion of *safety*, introduced in [1] to model contig assembly. Given a problem $\mathcal{P}$, we say that a partial solution to $\mathcal{P}$ is *safe* if it is common to all solutions to $\mathcal{P}$. Assuming that the real solution is among the solutions to our computational problem $\mathcal{P}$, then safe partial solutions are common also to the real solution, and therefore correct. We would like to report *all* such safe partial solutions, and in fact, it suffices to focus on all *maximal* safe partial solutions, namely those such that any other safe partial solution is contained in a maximal one. Safety has precursors in Bioinformatics (*reliable regions* in sequence alignments [2]), but also in combinatorial optimization (*persistency* in bipartite matchings [3]). Safety has also been applied to other genome assembly sub-problems such as gap filling [4].

The algorithmic toolkit for safety in genome assembly is quite developed by now. Indeed, a typical computational solution to a genome assembly problem is some type of walk in an assembly graph. For example, if we are sequencing a single bacteria and building an edge-centric de Bruijn graph from the reads, then the solution is a circular edge-covering walk [1]. A safe partial solution (i.e., contig) is thus a walk appearing as a subwalk of all such circular edge-covering walks. This graph-theoretic problem has been shown to admit efficient algorithms computing all their maximal safe walks [5], [6]. These run in time $O(|V||E|)$, where V and E are sets of vertices and edges, respectively, of the assembly

• *Manuel Cáceres, Massimo Cairo, and Alexandru I. Tomescu are with the Department of Computer Science, University of Helsinki, 00100 Helsinki, Finland. E-mail: {manuel.caceresreyes, alexandru.tomescu}@helsinki.fi, cairomassimo@gmail.com.*
• *Brendan Mumey is with the School of Computer Science, Montana State University, Bozeman, MT 59717 USA. E-mail: brendan.mumey@montana.edu.*
• *Edin Husić is with the Department of Mathematics, London School of Economics and Political Science, WC2A 2AE London, U.K. E-mail: e.husic@lse.ac.uk.*
• *Romeo Rizzi is with the Department of Computer Science, University of Verona, 37129 Verona, Italy. E-mail: romeo.rizzi@univr.it.*
• *Kristoffer Sahlin is with the Department of Mathematics, Science for Life Laboratory, Stockholm University, 106 91 Stockholm, Sweden. E-mail: ksahlin@math.su.se.*

Manuscript received 26 January 2021; revised 16 November 2021; accepted 23 November 2021. Date of publication 30 November 2021; date of current version 8 December 2022.
This work was partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 Research and Innovation Programme under Grant 851093 (SAFEBIO), by the Academy of Finland under Grants 322595 and 328877, by the U.S. Fulbright Program, the Fulbright Finland Foundation, the Helsinki Institute for Information Technology (HIIT), as well as the National Science Foundation under Grant DBI-1759522.
(Corresponding author: Manuel Cáceres.)
Digital Object Identifier no. 10.1109/TCBB.2021.3131203

graph. Recently, these results were generalized for a plethora of different genome assembly models in a common framework known as the *hydrostructure* [7].

Over the last decade, sequencing technologies have been applied to mixed settings, where a sample contains *multiple distinct* genomic sequences, that may differ in varying degrees. Genome assembly has thus been extended to *multi-assembly* problems [8] asking to reconstruct *all* such individual genomic sequences from mixed reads sequenced from all of them. Popular instances of such multi-assembly problems ask to reconstruct e.g., the RNA transcripts present in a cell population [8], or the quasi-species of a virus present in infected cells [9]. Although the of concept safety is implicitly used in current multi-assemblers, there exists a lack of formal treatment of this notion, despite its relevance and widespread adoption in the standard genome assembly problem.

1.1 Path Covers and Multi-Assembly

Given a directed acyclic graph G (or *DAG*, for short), we consider a *solution* to the multi-assembly problem to be a set of paths in G such that every vertex of G appears in at least one path. Such a set of paths is called a *path cover* of G, and it is present at the core of practical tools for both RNA transcript assembly and viral quasi-species assembly, as we review next.

A common graph used in the multi-assembly of RNA transcripts is a *splicing graph*, obtained by first identifying exons from the RNA read alignments to the reference. Every exon is then added as a vertex, and every read overlapping two exons indicates a possible *splicing junction* and is added as an edge between the two exons. The edges are directed "from left to right" (in the reference genome), making the graph a DAG.

An RNA transcript naturally corresponds to a path in such a DAG, thus the set of all RNA transcripts (solution to the RNA transcript assembly) corresponds to a path cover of the DAG. Various criteria have been proposed to define what path covers are actually a solution, e.g., those optimal with respect to some combinatorial optimization problem, or some statistical model. While state-of-the-art methods also include many practical steps and heuristics, they have one of two main approaches at their core. The first approach, used up to some variations and constraints by [10], [11], [12], [13], [14], define a solution as a path cover with a minimum number of paths (*minimum path cover*, or *MPC*), or with a number of paths smaller than an upper bound. The second approach, used up to some variations by [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], is based on finding a flow in the splicing graph best explaining the observed read coverage, and then on decomposing the flow into a small number of paths. Such paths also form a path cover, which is now optimal with respect to some flow criteria, not just cardinality.

Path cover models are also used in the assembly of reads sequenced from all viral quasi-species in a sample. Minimum path cover-like methods include [27], [28], and methods based on path covers optimal to some flow criteria include [29], [30].

1.2 Our Contributions

We give the first algorithms outputting all maximal safe paths for a natural notion of path cover, which we call *constrained* path cover. A constrained path cover can have at most a given

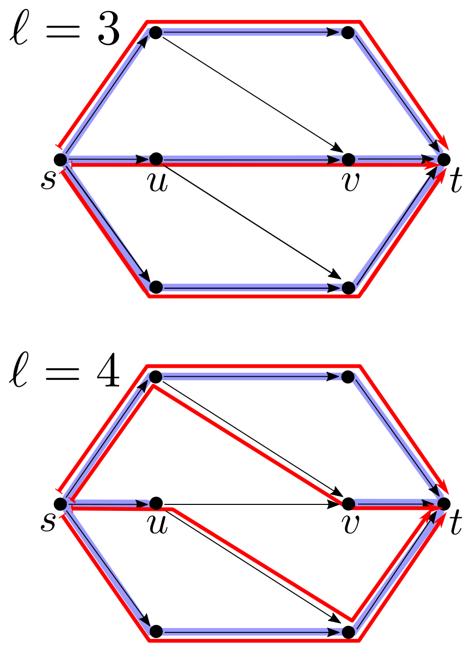

Fig. 1. An DAG G, safe paths (highlighted) and an example path cover (long arrows) for G, $S = \{s\}$, $T = \{t\}$, $\ell = 3$ (top) and $\ell = 4$ (bottom). The path cover $\mathcal{P}$ on the top is the unique constrained path cover for $\ell = 3$, therefore safe paths are exactly the paths of $\mathcal{P}$. The path cover on the bottom does not have the path $P = s, u, v, t$, therefore P is not safe for $\ell = 4$.

number $\ell \geq k = \mathsf{width}(G)$[1] of paths, and its paths are now required to start and end in given vertex sets, S and T, respectively. This notion generalizes that of an MPC,[2] which is a classical combinatorial object, dating back to Dilworth's and Fulkerson's results in the 1950s [31], [32], and appearing in standard textbooks such as [33]. More formally, given a DAG G, sets $S, T \subseteq V$, and an integer $\ell \geq k$, we say that a path P is *safe for* G, S, T, ℓ, if for every constrained path cover $\mathcal{P}$ of G, P is a *subpath* of some path of $\mathcal{P}$. When G, S, T and ℓ are clear from the context, we just say that P is safe. Fig. 1 illustrates these concepts.

Our safe algorithms are obtained using a general "avoid-and-test" approach. Intuitively, given a structure to be checked for safety, we transform the graph so that no solution can use the structure fully, while not changing the other properties of the graph. If a solution still exists, then it must avoid the structure, proving it is unsafe. This general approach dates back to finding edges present in all maximum matchings of a bipartite graph [3]. However, safe paths can contain more than one edge, which requires a more complex approach to "avoid-and-test".

We start with an arbitrary minimum-sized constrained path cover $\mathcal{P}$, and we test the safety of every subpath P of a path in $\mathcal{P}$. By definition, all safe paths are subpaths of some path of $\mathcal{P}$. We introduce the reduction of G with respect to P, G^P, and we show that P is a safe path if and only if $\mathsf{width}(G^P) > \ell$ and the subpath formed by excluding the last vertex of P is safe. A naive implementation of these ideas leads to a running time of $O(k|V|^2\mathsf{mpc}(G))$ for computing maximal safe paths, where $\mathsf{mpc}(G)$ denotes the running time

1. Given a DAG G, its *width*, denoted $\mathsf{width}(G)$, equals the minimum size of a path cover of G.

2. An MPC is a constrained path cover with $\ell = k$ and $S = T = V$.

of computing an MPC of G. However, we improve these running times by using two additional ingredients. First, since we need to output only *maximal* safe paths, it suffices to do a two-finger scan over each path of $\mathcal{P}$. The scan advances the right finger if the path P between them is safe, or the left finger otherwise. Second, to avoid recomputing an MPC of G^P for each different P, we build a *flow* data structure on top of G, which allows determining whether $\mathsf{width}(G^P) > \ell$ in time $O(\max(1, k + \mu - \ell)|E|)$, where μ is the number of paths of $\mathcal{P}$ containing the last edge of P. Our data structure is based on the concept of "shrinking", previously used to obtain efficient parameterized solutions for the problem of computing an MPC [34], [35]. The application of these two ideas improves the running time of our solution to $O(k^2|V||E|)$.

To test our solution and provide a proof-of-concept study highlighting the potential usefulness of safe paths in RNA transcript assembly, we consider splicing graphs constructed from transcript annotation of different species, including human. As such, the splicing graphs used in our experiments correspond to work in *perfect* conditions, removing the biases introduced by errors prior to the splicing graph construction [36]. We denote the maximal safe paths in this application as *RNA contigs*. As a comparison baseline, we also consider a basic notion of safe paths, namely those maximal paths whose internal vertices have in-degree and out-degree equal to one. These are called *unitigs* and are commonly used in genome assembly [37], [38], [39]. On these datasets, RNA contigs for constrained path covers are up to 8 times longer than unitigs, while being over 99% precise, and take less than 15 seconds to be found. Moreover, if we define the maximum relative coverage of a transcript as the length of the longest RNA contig segment inside it, divided by the length of the transcript, then transcripts have maximum relative coverage up to 80%. Moreover, RNA contigs cover over 70% of the coding sequences of transcripts in most cases.

We hope that our findings introduce a new toolkit and perspective on the notoriously hard transcript assembly problem. We envision that, after enhancements dealing with real data issues, safe paths could be applied into RNA assembly (or multi-assembly) pipelines as follows:

- By outputting only RNA contigs, in the same way as state-of-the-art genome assemblers output contigs. Our initial results support this, since RNA contigs, albeit not being full transcripts, still have significant length, and at the same time are correct (partial) transcript assembly results.
- As a preprocessing step to methods that extend given strings into full transcripts. For example, [12], [18] first conservatively assemble the RNA-Seq reads into some longer sequences, and use these to guide the (more heuristic) assembly of full RNA transcripts.
- By taking the RNA transcripts output by any existing RNA transcript assembler, and marking the maximal transcript substrings that match a safe path. In this way, our method could indicate some parts of an RNA assembly solution that are likely to be correct.

The rest of this paper is structured as follows. In Section 2 we develop the theory behind our algorithms reporting safe paths for constrained path covers. In Section 3 we discuss the RNA transcript assembly problem in more detail. We also present the experimental setup and results for RNA contigs, as an application of safe paths. We conclude the paper in Section 4.

2 Safe Paths for Constrained Path Covers

As previously discussed, a solution to a multi-assembly problem can be seen as a path cover of a DAG G. Therefore, safe paths are paths in G that are common to all path covers of G. However, this definition of safety is too permissive in formal terms, since it allows for $\mathcal{P} = \{(v) : v \in V\}$ (one path for each vertex v, consisting only of vertex v) to be a solution (path cover), thus yielding maximal safe paths to be isolated vertices.[3]

To overcome this problem, we restrict the solution space by including further practical information. We formalize this in the concept of a constrained path cover.

Definition 1 (Constrained path cover). *Let $G = (V, E)$ be a DAG, $S, T \subseteq V$ be two sets of vertices such that $sources(G) \subseteq S$, $sinks(G) \subseteq T$,[4] and let $\ell \geq \mathsf{width}(G)$. We say that a path cover $\mathcal{P}$ of G is a* constrained path cover *(for G, S, T, ℓ) if:*

(a) Every path $P \in \mathcal{P}$ starts at some vertex in S and ends at some vertex in T.
(b) The number of paths of $\mathcal{P}$ is at most ℓ, $|\mathcal{P}| \leq \ell$.

Constraint *(a)* restricts the paths of $\mathcal{P}$ to start and end with some of the vertices of given sets of vertices S and T, respectively. We say that the paths of $\mathcal{P}$ go from S to T. This restriction incorporates the fact that in some contexts, such as RNA transcript assembly, some candidates of the starting / ending vertices may be identified, e.g., as those vertices whose read coverage present an initial upward / final downward slope [21], [22]. On the other hand, constraint *(b)* restricts the solution to contain at most ℓ paths, which is used to relax the minimality condition present in minimum path cover-like multi-assemblers as previously reviewed.

Note that, if $S = T = V$ and $\ell = \infty$, then we remove both restrictions and recover the classical definition of path cover. Besides, if instead $\ell = k$, then the constrained path covers correspond to the MPCs of G.

The conditions $sources(G) \subseteq S, sinks(G) \subseteq T$ are necessary to ensure that at least one constrained path cover exists (otherwise there exists a vertex that is not reached from S or does not reaches T, which cannot be covered). On the other hand, the condition $\ell \geq \mathsf{width}(G)$ is also necessary, since a constrained path cover is, in particular, a (classical) path cover of G, thus of size at least $\mathsf{width}(G)$.

Moreover, these three conditions (on S, T and ℓ) combined suffice to ensure that at least one constrained path cover exists, which follows from the following clemma.

Lemma 1. *Let G be a DAG, $S, T \subseteq V$ be two sets of vertices such that $sources(G) \subseteq S, sinks(G) \subseteq T$. Then, the minimum number of paths of a path cover with paths from S to T is $\mathsf{width}(G)$.*

3. In particular, maximal safe path must be subpaths of $\mathcal{P} = V$, thus isolated vertices.

4. $sources(G)/sinks(G)$ are the vertices of in/out-degree zero.

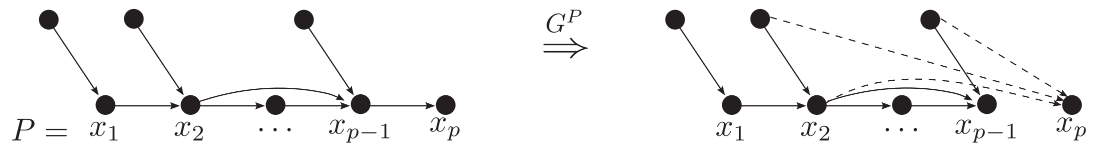

Fig. 2. The construction G^P from Definition 2, for $P = x_1, \ldots, x_p$: the edge (x_{p-1}, x_p) is removed, and (transitive) edges are added from the in-neighbors (except from the path predecessor) of the path vertices (except x_1) to x_p, shown as dashed.

The proof of Lemma 1 (see Supplemental material 1, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2021.3131203, for the proofs of our lemmas and theorems) also give us a way to compute our initial solution: we first compute an MPC of G and then extend their paths to S and T.

Next, we observe that it is sufficient to consider $|S| = |T| = 1$ by using the following reduction. We build the graph $G' = (V', E')$, such that, $V' = V \cup \{s, t\}$ (with $\{s, t\} \cap V = \emptyset$), and $E' = E \cup \{(s, u), u \in S\} \cup \{(v, t), v \in T\}$. That is, we add a new unique source vertex pointing to every vertex of S, and a new unique sink vertex pointed from every vertex of T. Note that, there is a one-to-one correspondence between constrained path covers for G, S, T, ℓ and constrained path covers for $G', \{s\}, \{t\}, \ell$ (or just G', s, t, ℓ), since we can obtain one from the other by adding/removing s and t to/from the paths. As such, we can reduce the computation of safe paths for constrained path covers for G, S, T, ℓ (or just safe paths for G, S, T, ℓ) by computing safe paths for G, s, t, ℓ (we abuse of the notation and use G instead of G' for simplicity).

To decide whether a path P is safe (for G, s, t, ℓ) we introduce the construction G^P, which we call *the reduction of G with respect to P*.

Definition 2. *Given a DAG $G = (V, E)$ and a path $P = x_1, \ldots, x_p$ of G, we define $G^P = (V, E^P)$, where*

$$E^P = (E \setminus \{(x_{p-1}, x_p)\}) \cup \bigcup_{i=2}^{p} \{(u, x_p) \mid u \in N^-(x_i) \setminus \{x_{i-1}\}\}.$$

Fig. 2 illustrates G^P. The main idea behind this construction is that no path of G^P can contain P entirely, since (x_{p-1}, x_p) is removed. However, the usage of every proper suffix of P, except if this is used as the beginning of a path, is emulated by the *transitive edges* drawn from the in-neighbors of the vertices of the paths (except from the previous vertex on the path, and not for the first vertex) to x_p. Proper suffixes of P (except x_p) used as the beginning of a path are not emulated in G^P, and adding transitive edges from arbitrary vertices of P to represent these suffixes will ultimately emulate P. To solve this issue we incorporate to our hypothesis the safety of subpaths of P. Specifically, we obtain the following theorem.

Theorem 2. *Let $G = (V, E)$ be a DAG, $s \in V$ the unique source of G, $t \in V$ the unique sink of G, and let $P = x_1, \ldots, x_p$ be a proper path of G, such that $x_1, \ldots, x_{p-1}$ is a safe path for G, s, t, ℓ. Then we have one of two cases:*

1) *$x_{p-1} \in sinks(G^P) \vee x_p \in sources(G^P)$, in which case P is a safe path for G, s, t, ℓ.*
2) *otherwise, P is a safe path for G, s, t, ℓ if and only if $\mathsf{width}(G^P) > \ell$.*

Our first algorithm for computing maximal safe paths (for G, s, t, ℓ) uses Theorem 2 to test the safety of each of the $O(k|V|^2)$ subpaths of the paths of an initial minimum-sized constrained path cover $\mathcal{P}$ of G ($O(|V|^2)$ subpaths for each of the k (by Lemma 1) paths of length $O(|V|)$ each). We process these subpaths in increasing order of their lengths. When processing a subpath $P = x_1, \ldots, x_p$ we inductively know whether $x_1, \ldots, x_{p-1}$ is safe, and we can proceed accordingly. If $x_1, \ldots, x_{p-1}$ is not safe then P is not safe either (by definition of safety). Otherwise, if $x_1, \ldots, x_{p-1}$ is safe we use Theorem 2 to test whether P is safe. We build G^P, if $x_{p-1} \in sinks(G^P) \vee x_p \in sources(G^P)$ we report P as safe, otherwise, we compute an MPC $\mathcal{P}'$ of G^P and report P as safe if and only if $|\mathcal{P}'| = \mathsf{width}(G^P) > \ell$. If P is safe we add it to a set of maximal safe paths and remove $x_1, \ldots, x_{p-1}$ and $x_2, \ldots, x_p$ from this set since P invalidates their maximality. Note that with this rule every maximal safe path is captured and every non-maximal safe path is removed at some point. We call this approach *unoptimized*, and it runs in time $O(k|V|^2\mathsf{mpc}(G))$, where $\mathsf{mpc}(G)$ denotes the running time of computing an MPC.

By running a *two-finger* algorithm on every path P of $\mathcal{P}$, we can compute the *maximal* safe paths inside P in time $O(|P|) = O(|V|)$. We maintain pointers x and y on vertices of P, and the invariant that the subpath of P between x and y is a safe path. First, we try to extend this subpath by moving y to the next vertex of P. Since we know that our current subpath is safe we use Theorem 2 (as previously explained) to test if the extension is safe. If it is not, we move x (and also y, if $x = y$) to the next vertex of P and try again. Maximal safe paths reported by this approach are only guaranteed to be maximal within P (the path from which they were reported), but not necessarily maximal in the whole G. To efficiently report all maximal safe paths of G, we collect the maximal safe paths inside each $P \in \mathcal{P}$, and then filter those that are subpaths of already reported paths. For this we apply the generalized suffix tree approach used in Step 3 of [40]. This lets us filter the paths contained in another path in total time $O(N)$ where N is the sum of the lengths of all paths. Here, $N = O(k|V|^2)$, since the two-finger algorithm reports $O(|V|)$ safe paths each of length $O(|V|)$, for each path the k paths of $\mathcal{P}$. As such, this approach runs in time $O(k|V|^2 + k|V|\mathsf{mpc}(G)) = O(k|V|\mathsf{mpc}(G))$.

Our last optimization consists in speeding up the second case of the safety test of Theorem 2. Each such test involves computing an MPC of G^P, taking $\mathsf{mpc}(G)$ time. Instead, we use our initial solution $\mathcal{P}$ to compute a *flow* data structure at the beginning of the algorithm. This data structure takes additional $O(k|V| + |E|)$ time to construct, but allows us to test whether $\mathsf{width}(G^P) > \ell$ in time $O(\max(1, k + \mu - \ell)|E|)$, where μ is the number of paths of $\mathcal{P}$ containing the last edge of P. In broad terms, the data structure corresponds to a reduction of path cover to a *flow* network, which has been used before for finding an MPC [34], [35], [41], [42], [43], and the answer to the queries "$\mathsf{width}(G^P) > \ell$?" are based on the concept of *shrinking* of a flow in this flow reduction [34], [35].

Lemma 3. *Let $G = (V, E)$ be a DAG of width k, and $\mathcal{P}$ an MPC of G. We can build a data structure in time $O(k|V| + |E|)$, answering whether $\mathsf{width}(G^P) > \ell$ in time $O(\max(1, k + \mu - \ell)|E|)$, for every $\ell \geq k, P = x_1, \ldots, x_p$ proper path of G, with $\mu = |\{P \in \mathcal{P} \,|\, (x_{p-1}, x_p) \in \mathcal{P}\}|$.*

The proof of Lemma 3 (see Supplemental material 1, available online) shows us that if we are in the second case of Theorem 2, then $\mathsf{width}(G^P) \leq k + \mu \leq 2k$. As such, if $\ell \geq 2k$, then $\mathsf{width}(G^P) > \ell$ will always be false.

Observation 4. If $\ell \geq 2k$, safe paths for G, s, t, ℓ are the same as safe paths for G, s, t, ∞.

Using the two-finger algorithm and the flow data structure, we obtain the main theoretical result of this paper.

Theorem 5 (Safe paths for constrained path covers). *The maximal safe paths for constrained path covers (for an input $G = (V, E)$, s, t, ℓ), can be computed in time $O(\max(1, 2k - \ell)k|V||E|)$, where $k = \mathsf{width}(G)$.*

3 Application to RNA Transcript Assembly

The functioning of the cell is based on the transcription of genes into transcripts, followed by the translation of the transcripts into proteins. This makes the set of transcripts present in a cell (the *transcriptome*) an important link between DNA and phenotype, and can give information of the current and future state of a cell. High-throughput sequencing of transcripts (RNA-seq) started in 2008 [44], [45], and later proved essential in characterizing gene regulation and function, development and diseases, including cancer [46], [47], [48], [49]. In complex organisms, one gene can produce different transcripts, each in a different number of copies. For example, about 95% of multi-exon genes in humans produce multiple transcripts through alternative splicing [50]. Alternative splicing alters the set of exons transcribed by the gene, but different transcripts can still share exons.

The transcript assembly problem aims to reconstruct the set of transcripts present in a set of RNA-seq reads. While long reads hold the potential to sequence through the full transcripts, thus resolving the full transcript structure, there are inherent biases in the protocols which makes them unable to sequence longer transcripts [51], [52]. In addition, long-read alignment software have been shown to produce inaccurate alignments [53], [54] on which the assembly methods rely on. Moreover, current state-of-the-art long-read transcript assembly methods are still in active development, with precision below 50% on biological data [55]. Due to these limitations, short-read RNA-seq assembly remains a viable option.

While the transcript assembly problem has attracted great interest from the community, with a proliferation of methods proposed [10], [11], [15], [16], [17], [18], [19], [20], [23], [56], [57], [58], [59], [60], [61], assembling RNA-seq reads remains a challenge, with RNA assembly methods having a precision under 50%-60% on human data [18], [62]. In addition, current algorithms that aim to produce full-length transcripts employ various heuristics and thresholds to increase contiguity of the transcript under assembly, which makes results vary significantly in quality with different parameter settings [63].

The importance of having *reliable* transcript assembly results is further underlined by two recent related works [64], [65]. However, they tackle the reliability issue with a significantly different approach, as they provide methods to calculate a "confidence" range for the abundance in the sample of a given candidate transcript. In Fig. 3 we show an example of our safety approach applied to an RNA splicing graph of the human transcriptome built from genome annotation. We call RNA contigs the maximal safe path in this context.

3.1 Implementation and Experiments

We aim to motivate the use of safe paths as an alternative construct to unitigs for transcript assembly. Therefore, the main purpose of our experiments is to evaluate the construction time, correctness, and length of sequences produced by maximal safe paths compared to the sequences produced by unitigs. We compare the two approaches for RNA of real organisms, but avoid external artifacts introduced by aligning the RNA-seq reads, unsequenced regions, inferring exons from alignments, and constructing the splicing graph, which introduces many biases [36]. We instead build the splicing graph directly from gene annotation As such, the splicing graph used in our experiments corresponds to work in *perfect* conditions. Removing the biases introduced by errors prior to the splicing graph construction provides us a preliminary evaluation of RNA contigs in the transcript assembly problem. It is also worth noting that our datasets (annotated transcripts) as well as our algorithm are read coverage agnostics. Since the main objective of safe paths is to output information about the transcripts but not about their abundances, we do not consider this characteristic in our evaluation.

Datasets and Input. We started from gene annotation, by considering all transcript annotation from the Ensembl database [66] for different species as detailed in Table 1.

In case two exons $E = E_1E_2$ and $E' = E_2E_3$ from different transcripts have a suffix-prefix overlap E_2, we replace each occurrence of E in a transcript with E_1, E_2, and similarly for E'. This is common in splicing graph construction (see e.g., [17], [67]); the resulting exons are usually called *pseudo-exons*, but for simplicity we refer to them as exons. Graph vertices correspond to annotated (pseudo-)exons, and edges correspond to exons consecutive in some annotated transcript. Each weakly connected component of this graph is a splicing graph. For each dataset we considered all transcripts on the forward strand and build the corresponding splicing graphs. We also store the coding sequences on each transcript annotated by the corresponding Ensembl dataset. We say that a splicing graph is a *trivial instance* if it is formed by less than 3 exons or less than 2 transcripts. We filter out trivial graph instances from the dataset for our experiments.

To correlate the results with the complexity of the data, we partitioned all the annotated transcripts based on their

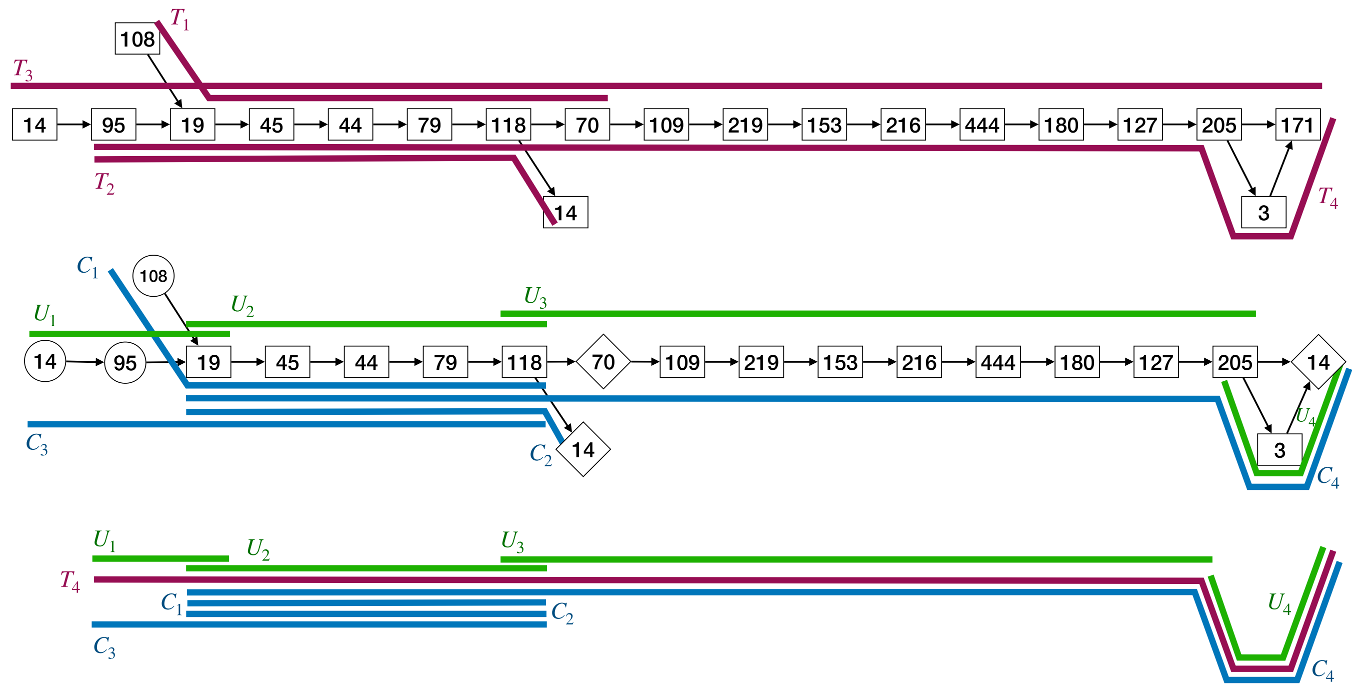

Fig. 3. *Top*: A splicing graph built from four annotated transcripts from human chr 18; vertex labels denote exon lengths. The transcripts form a constrained path cover with $\ell = 4$. *Middle*: The ST-unitigs of the graph, and the maximal safe paths (i.e., RNA contigs) w.r.t. constrained path covers of it, for $\ell = 4$, and S and T being the set of start exons (as circle vertices) and end exons (as diamond vertices), respectively, of the transcripts. For unitig U_2, contained in RNA contig C_4 we have a relative improvement of 3.2× exons, and of 7.2× bases. *Bottom*: Transcript T_4 and all the longest RNA contig segments inside it. For its unitig segments, the maximum coverage is 10 exons, and 1841 bases. For its RNA contig segments, the maximum coverage is larger by 6 exons, and by 361 bases. See Section 3.1 for definitions of these metrics.

TABLE 1
Datasets/Species Considered in Our Experiments: Two Mammals (Including Human), Two Plants, One Insect, and One Fungus

Dataset	Assembly	Transcripts	Coding sequences	Splicing graphs
Homo sapiens (Human)	GRCh38.p13	104552	52711	11337
Mus musculus (Mouse)	GRCm39	55281	31593	10346
Triticum aestivum (Wheat)	IWGSC	21225	21141	8468
Hordeum vulgare (Barley)	IBSC v2	111695	106219	12835
Drosophila melanogaster (Fruit fly)	BDGP6.32	12815	11585	3608
Magnaporthe oryzae (Rice blast)	MG8	407	365	196

The table also shows the number of transcripts, coding sequences and splicing graphs after filtering trivial *graph instances of the problem (see Section 3.1).*

length, into *small* (1-2000 bases), *medium* (2001-5000 bases) and *large* ($>$ 5000 bases). We also partitioned coding sequences (1-1000, 1001-2500, $>$ 2500 bases) and splicing graphs (3-15, 16-50, $>$ 50 vertices) in these three categories.[5]

Implementations. For each splicing graph, we fixed the sets S and T as the set of exons appearing as first, or as last, respectively, in some annotated transcript. We assume that such S and T can be detected at the splicing graph construction phase.[6] In practice, this could be done as stated in Section 2. The algorithm from Theorem 5 (optimized) was implemented in C++, and uses an implementation of the Edmons-Karp algorithm [68], [69] from the LEMON graph library [70] for theoretical guarantees. To implement the unoptimized version (not two-fingers, nor the flow data structure) we used the LEMON's implementation of the Network simplex algorithm [71], [72] (for better performance). The input data manipulation and evaluation code was written in Python, including the computation of the ST-unitigs discussed later. Our entire code and input datasets are publicly available at https://github.com/algbio/SafePathsRNAPC. Our C++ code was compiled with optimization flag `-O9` using compiler `gcc` version `8.3.0`. The experiments ran on a single thread in a Linux machine (Debian GNU/Linux 10, kernel `4.19.0-10-amd64`), with processor Intel (R) Core (TM) i7-8750H @ 2.2 GHz and 16 GB of RAM.

Choice of Parameter ℓ and its Effect on Correctness. For a splicing graph, let us denote by t the number of true transcripts. By construction and definition, the RNA transcripts of a splicing graph correspond to a constrained path cover with at most $\ell = t$ paths. Thus, safe paths for such constrained path covers are also *correct*, in the sense that they appear in the true RNA transcripts (because they appear in any constrained path cover with at most $\ell = t$ paths). However, safe paths for constrained path covers with $\ell < t$ may

5. The grouping was made so that approximately matches the 60% smallest part of the data (small) the next 30% (medium) and the remaining 10% largest part of the data.

6. This assumption is also part of the *perfect* conditions in which we ran our experiments.

not be correct (i.e., may not appear in some true RNA transcript). As such, for each splicing graph, we first compute the smallest size (i.e., number of paths) of a constrained path cover, for the fixed S and T, and we denote this size by k. Since t is unknown in practice, we perform experiments for $\ell \in \{k, k+1, \dots, 2k\}$, to evaluate which is a good choice for the parameter ℓ. Note that maximal safe paths for $\ell = 2k$ are common to all path covers (of any size) as stated in Observation 4, thus if $t > 2k$ we can interpret the results for $\ell = t$ using the results for $\ell = 2k$ (even when we did not run the algorithm for $\ell = t$).

Baseline Comparison With ST-Unitigs. We also implemented a standard strategy used by genome assemblers to compute contigs. This involves reporting those paths whose internal vertices have in-degree and out-degree equal to 1 (and with at least one internal vertex), also called *unitigs* [37]. If no vertex of S or of T appears in a unitig, then the unitig is safe for constrained path covers, for any ℓ. Intuitively, to cover an internal vertex v of a unitig P (which exists by definition), one must arrive from some vertex of S, then entirely traverse the prefix of P until v, and then arrive to some vertex in T, thus entirely traverse the suffix P from v. However, if e.g., some vertex of T appears in P after v, the path in the constrained path cover covering v may stop before reaching the end of P. As such, we say that an *ST-unitig* is a unitig containing no vertex of S and T as internal. It holds that ST-unitigs are safe and correct for constrained path covers, for any ℓ.

Evaluation Metrics. To evaluate the performance of RNA contig we report several metrics computed in *vertex length*, that is, number of exons, and in *base length*, that is, the total number of bases in all the exons considered.

Our first two metrics show the improvement of RNA contigs with respect to ST-unitigs. Since ST-unitigs must be covered by one path, they are subpaths of some maximal safe path. As such, for every ST-unitig we compute its *improvement* as the longest RNA contig containing it, and report the difference and ratio of their lengths.

From this point onward, by *contig* we denote both an RNA contig, and an ST-unitig. Our second set of metrics measure the sensitivity of both approaches from different perspectives. First, for every transcript (and every approach) we compute the longest contig *segment* inside the transcript, that is, the longest path that is a common subpath of the transcript and of some contig. We say that the length of this subpath is the *maximum coverage* of the transcript. To specifically measure the coverage of coding sequences, we also compute their maximum coverage, and we call it *maximum coding coverage.* Besides, for every transcript we compute a standard metric used in genome assembly, namely, the *e-size* [73] of the transcript. In this case, the e-size is the average length of all contigs inside the transcript overlapping a random location of it. More precisely, for every position in a transcript T, we take the average length over all contig segments inside T overlapping that position. The e-size of the transcript is obtained as the average of such averages, over all positions of the transcript. Finally, to normalize our results, we report our sensitivity metrics divided by the length of the corresponding transcript.

Our final metric computes the *precision* of contigs per splicing graph. We classify a contig as *correct* if it is a subpath of at least one annotated transcript. The precision over a splicing graph is the total length of the correct contigs, divided by the total contig length. Concrete examples of some of these metrics can be found in Fig. 3.

Safe Paths of a Variation Graph. To demonstrate the time scalability of our algorithms, we computed safe paths of a variation graph common to all path covers ($\ell = \infty$). Note that this is not related to multi-assembly, the main application envisioned by this paper. However, safe paths in this context can be interpreted as genomic regions common to all individuals represented by the variation graph. We used a variation graph built from the Leukocyte Receptor Complex (LRC) [74], which constitutes one of the most diverse variant spots. We extracted subgraphs of different sizes of the LRC variation graph and run the optimized and unoptimized versions of our algorithm. Sugraphs were taken as subgraphs induced by a consecutive segment of vertices in a topological order, which guarantees the width of the resulting subgraph to be at most the width of the original graph [75].

3.2 Results

In this section we show a summary of the results of the metrics described in Section 3.1 across the different datasets. A detailed breakdown of the metrics and running times for each dataset can be found in the Supplemental material 2, available online. For the metrics maximum coverage, e-size and precision, we only show them in base length. We observed that average precision for $\ell = k$ is over 75% for every dataset except barley (over 60%) and rice blast (over 44%). On the other hand, the average precision for $\ell = k+1$ is over 99% for all species and 100% for $\ell = t$ and $\ell = 2k$ as expected. Moreover, in general, all metrics are very close (less than 1% of difference at most) for $\ell = k+1$, $\ell = t$ and $\ell = 2k$, suggesting that safe paths derived with *prior* knowledge of the number of transcripts ($\ell = t$) can be reasonably approximated as safe paths for $\ell = k+1$ or safe paths common to *all* path covers ($\ell = 2k$), with a small *tradeoff* of precision versus coverage. In the next results we fix $\ell = k+1$ for RNA contigs.

Table 2 shows the improvement of RNA contigs with respect to ST-unitigs. Large graphs have longer RNA contigs, suggesting that maximal safe paths manage to connect paths in different sectors that ST-unitigs are not able to capture with their simple rule. Overall, RNA contigs are from 1.4 (small graphs, human) to 8.1 (large graphs, wheat) times longer in terms of bases, and from 1.17 to 5.6 times longer in terms of vertices (i.e., exons). The most improvement with respect to ST-unitigs occurs in wheat and fruit fly, whereas the least improvement in human, mouse and rice blast, the latter with just a few graphs (see Table 1) with no more than 15 vertices each.

In Table 3 we show e-size and maximum coverage (normalized by transcript length) at transcript level, and in Table 4 we show them at splicing graph level. In terms of precision (Table 4), as discussed before, RNA contigs reach over 99% on average for $\ell = k+1$ for all datasets and size of splicing graphs, while ST-unitigs are 100% precise as expected (as well as RNA contigs for $\ell = 2k$, see Supplemental material 2, available online). At transcript level (Table 3) e-size of RNA contigs is over 40% for human, meaning that at a random position on the transcript, the average length of

TABLE 2
Absolute (First Line) and Relative (Second Line) Improvement of RNA Contigs ($\ell = k + 1$) Over ST-Unitigs, in Terms of Base and Vertex Lengths

	small (3-15 vertices)		medium (16-50 vertices)		large (51-725 vertices)	
Dataset	bases	vertices	bases	vertices	bases	vertices
Human	160.95 1.41×	0.58 1.18×	245.65 1.80×	1.53 1.45×	233.60 1.96×	1.93 1.55×
Mouse	202.29 1.47×	0.72 1.23×	334.33 1.95×	2.02 1.57×	425.96 2.30×	3.13 1.81×
Wheat	765.49 2.84×	3.29 1.96×	1078.49 4.07×	6.74 2.81×	3103.23 8.14×	19.84 5.62×
Barley	150.79 1.89×	0.64 1.21×	192.70 2.56×	1.42 1.44×	162.73 2.91×	2.05 1.62×
Fruit fly	662.90 2.09×	1.59 1.48×	1339.42 3.20×	3.61 2.07×	3049.40 5.33×	5.97 2.67×
Rice blast	281.37 1.57×	0.55 1.17×	–	–	–	–

Values are averages over all ST-unitigs in the respective dataset and group.

TABLE 3
Metrics e-Size and Maximum Coverage, Divided by Transcript Length, for RNA Contigs ($\ell = k + 1$, First Row) and ST-Unitigs (Second Row), in Base Length

	small (1-2000 bases)		medium (2001-5000 bases)		large (5001-205012 bases)	
Dataset	`e-size`	`max-cov`	`e-size`	`max-cov`	`e-size`	`max-cov`
Human	0.49 0.33	0.63 0.49	0.41 0.33	0.56 0.48	0.43 0.37	0.58 0.52
Mouse	0.58 0.40	0.72 0.55	0.51 0.37	0.66 0.51	0.50 0.41	0.64 0.55
Wheat	0.72 0.49	0.81 0.63	0.64 0.46	0.74 0.60	0.63 0.49	0.74 0.62
Barley	0.43 0.35	0.57 0.48	0.28 0.23	0.42 0.37	0.22 0.19	0.37 0.33
Fruit fly	0.83 0.66	0.90 0.74	0.70 0.55	0.80 0.67	0.58 0.46	0.70 0.60
Rice blast	0.88 0.69	0.94 0.76	0.89 0.77	0.94 0.84	0.72 0.47	0.79 0.56

Values are averages over all annotated transcripts in the respective dataset and group.

TABLE 4
Precision (`prec`) and, e-Size and Maximum Coverage, Divided by Transcript Length, for RNA Contigs ($\ell = k + 1$, First Row) and ST-Unitigs (Second Row), Per Splicing Graph, in Base Length

	small (3-15 vertices)			medium (16-50 vertices)			large (51-725 vertices)		
Dataset	`prec`	`e-size`	`max-cov`	`prec`	`e-size`	`max-cov`	`prec`	`e-size`	`max-cov`
Human	1.00 1.00	0.68 0.49	0.81 0.62	0.99 1.00	0.48 0.35	0.63 0.51	0.99 1.00	0.37 0.27	0.52 0.42
Mouse	1.00 1.00	0.71 0.51	0.83 0.64	0.99 1.00	0.51 0.36	0.66 0.51	0.99 1.00	0.44 0.29	0.58 0.44
Wheat	1.00 1.00	0.72 0.49	0.80 0.63	1.00 1.00	0.58 0.44	0.70 0.58	1.00 1.00	0.51 0.40	0.62 0.52
Barley	0.99 1.00	0.68 0.54	0.80 0.66	0.99 1.00	0.39 0.33	0.54 0.47	0.99 1.00	0.23 0.18	0.36 0.32
Fruit fly	1.00 1.00	0.81 0.65	0.89 0.74	1.00 1.00	0.51 0.38	0.65 0.54	1.00 1.00	0.50 0.41	0.67 0.58
Rice blast	1.00 1.00	0.88 0.71	0.93 0.79	–	–	–	–	–	–

Values are averages over all splicing graphs in the respective dataset and group.

TABLE 5
Metric Maximum Coding Coverage, Divided by Transcript Length, for RNA Contigs ($\ell = k+1$, First Row) and ST-Unitigs (Second Row), in Base Length

Dataset	small (1-1000 bases)	medium (1001-2500 bases)	large (2501-40620 bases)
Human	0.70 0.59	0.54 0.45	0.48 0.41
Mouse	0.79 0.69	0.67 0.56	0.57 0.49
Wheat	0.85 0.67	0.83 0.68	0.77 0.65
Barley	0.70 0.59	0.56 0.49	0.50 0.43
Fruit fly	0.94 0.79	0.90 0.77	0.80 0.68
Rice blast	0.95 0.80	0.98 0.91	0.96 0.82

Values are averages over all coding sequences in the respective dataset and group.

all the safe "stretches" inside that transcript, and overlapping that location, is about 40% of the length of the transcript length, whereas for wheat and rice blast the e-size is over 60% and 70% respectively and only 22% for barley. At graph level, on small graphs, the longest RNA contig of a transcript is on average over 80% of the transcript length in all species. Table 5 shows maximum coding coverage (normalized by coding sequence length) at coding sequence level. RNA contigs cover over 70% of the coding sequences of transcripts in most cases. and over 48% in general. In terms of these global metrics, RNA contigs are typically between 10-20 percentage points over ST-unitigs. These results suggest that RNA contigs can provide significantly long and correct information about the RNA transcripts of a splicing graph, without any heuristic or complex assembly model.

Finding ST-unitigs takes less than 2 seconds, whereas RNA contigs are reported in 32 seconds for the unoptimized variant, and less than 12 seconds for the optimized in the slowest dataset (human, see Table 6). As such, the unoptimized algorithm, being the simplest, is a good initial candidate to be adapted and extended by future practical RNA contig assemblers. The difference of running times between both algorithms becomes clear when working in the subgraph of the variation graph LRC (see Table 7). When considering a subgraph of 10000 vertices the optimized version takes less than two minutes while the unoptimized more than 20 minutes. Finally, for the subgraph of 50000 vertices the optimized version takes less than 1 hour whereas the unoptimized runs in more than 17 hours.

TABLE 6
Running Times for Our Two Implementations Finding all RNA Contigs in All Splicing Graphs, for constrained Path Covers With $\ell = k+1$

Dataset	Unoptimized (secs)	Optimized (secs)
Human	32.55	11.29
Mouse	12.34	4.78
Wheat	2.88	0.82
Barley	29.13	10.95
Fruit fly	0.52	0.30
Rice blast	0.01	0.01

The Unoptimized column corresponds to the algorithm not using the two-finger approach nor the flow data structure. The Optimized column corresponds to our algorithm using these optimizations.

TABLE 7
Running Times for Our Two Implementations of Maximal Safe Paths ($\ell = \infty$) on Subgraphs of a Variation Graph Built From the Leukocyte Receptor Complex (LRC)

Graph	Unoptimized (hh:mm:ss)	Optimized (hh:mm:ss)
LRC 10000	00:22:27	00:01:52
LRC 20000	02:25:33	00:10:38
LRC 30000	03:59:03	00:20:18
LRC 40000	10:57:50	00:35:10
LRC 50000	17:45:48	00:57:05

The number in the graph name indicates the number of vertices in the subgraph.

4 CONCLUSION

The results of this paper are two-fold. On the theory side, we considered a natural generalization of the classical problem of minimum path cover, including more practical constraints, which we called constrained path covers. We devised the first algorithms finding all maximal safe paths for them, through a general "avoid-and-test" framework, that could have applications in other safety problems. We showed how the direct versions of these algorithms can be improved by reusing computation, with the help of a flow data structure and a two-finger computation of maximal safe paths.

On the practical side, we implemented our algorithms and offer these implementations in publicly available repositories for practitioners and further development of our ideas. As an application of our algorithmic ideas, we proposed safe paths for constrained paths covers as a contig model in RNA transcript assembly.

We evaluated the benefits of RNA contigs on transcript annotation (*perfect* conditions) of various species and observed for the first time that RNA contigs contain significantly long parts of the transcripts. This fact was not obvious from the outset, because in practice the problems may admit many different path covers, which have very little subpaths in common, and thus very short safe paths overall. However, our results show that RNA contig assembly is indeed a valid approach in transcriptome assembly. We also provided several key computational techniques that can lay at the core of future practical tools. As such, once RNA contigs are possibly enhanced with heuristics (for example from [12], [16], [18], [20], [57]), RNA contig assembly could enjoy the same success as contig assembly does in genome assembly.

Finally, our results are not limited to RNA transcript assembly but to any multi-assembly problem whose solution can be modeled with a path cover. For example, in the assembly of reads sequenced from all *viral quasi-species* in a sample there exist minimum path cover-like methods [27], [28], and methods based on path covers optimal to some flow criteria [29], [30].

Acknowledgments

We thank the anonymous reviewers for their useful suggestions to improve this work.

References

[1] A. I. Tomescu and P. Medvedev, "Safe and complete contig assembly through omnitigs," *J. Comput. Biol.*, vol. 24, no. 6, pp. 590–602, 2017.

[2] M. Vingron and P. Argos, "Determination of reliable regions in protein sequence alignments," *Protein Eng. Des. Selection*, vol. 3, no. 7, pp. 565–569, 1990.

[3] M.-C. Costa, "Persistency in maximum cardinality bipartite matchings," *Oper. Res. Lett.*, vol. 15, no. 3, pp. 143–149, 1994.

[4] L. Salmela and A. I. Tomescu, "Safely filling gaps with partial solutions common to all solutions," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 16, no. 2, pp. 617–626, Mar./Apr. 2018.

[5] M. Cairo, P. Medvedev, N. O. Acosta, R. Rizzi, and A. I. Tomescu, "An optimal $O(nm)$ algorithm for enumerating all walks common to all closed edge-covering walks of a graph," *ACM Trans. Algorithms*, vol. 15, no. 4, pp. 1–17, 2019.

[6] M. Cairo, R. Rizzi, A. I. Tomescu, and E. C. Zirondelli, "Genome assembly, from practice to theory: Safe, complete and linear-time," in *Proc. 48th Int. Colloq. Automata Lang. Program.*, 2021, pp. 43:1–43:18.

[7] M. Cairo, S. Khan, R. Rizzi, S. Schmidt, A. I. Tomescu, and E. C. Zirondelli, "Genome assembly, a universal theoretical framework: Unifying and generalizing the safe and complete algorithms," 2020, *arXiv:2011.12635*.

[8] Y. Xing, A. Resch, and C. Lee, "The multiassembly problem: Reconstructing multiple transcript isoforms from EST fragment mixtures," *Genome Res.*, vol. 14, no. 3, pp. 426–441, 2004.

[9] N. Beerenwinkel, H. Günthard, V. Roth, and K. Metzner, "Challenges and opportunities in estimating viral genetic diversity from next-generation sequencing data," *Front. Microbiol.*, vol. 3, Sep. 2012, Art. no. 329.

[10] C. Trapnell *et al.*, "Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and Isoform switching during cell differentiation," *Nat. Biotechnol.*, vol. 28, no. 5, 2010, Art. no. 511.

[11] L. Song and L. Florea, "CLASS: Constrained transcript assembly of RNA-Seq reads," *BMC Bioinf.*, vol. 14, no. 5, pp. 1–8, 2013.

[12] 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.

[13] J. Liu, T. Yu, T. Jiang, and G. Li, "TransComb: Genome-guided transcriptome assembly via combing junctions in splicing graphs," *Genome Biol.*, vol. 17, no. 1, pp. 1–9, 2016.

[14] 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 Res.*, vol. 30, no. 8, pp. 1181–1190, 2020.

[15] J. Feng, W. Li, and T. Jiang, "Inference of Isoforms from short sequence reads," *J. Comput. Biol.*, vol. 18, no. 3, pp. 305–321, 2011.

[16] W. Li, J. Feng, and T. Jiang, "IsoLasso: A LASSO regression approach to RNA-Seq based transcriptome assembly," *J. Comput. Biol.*, vol. 18, no. 11, pp. 1693–1707, 2011.

[17] S. Mangul, A. Caciula, S. Al Seesi, D. Brinza, A. R. Banday, and R. Kanadia, "An integer programming approach to novel transcript reconstruction from paired-end RNA-Seq reads," in *Proc. ACM Conf. Bioinf. Comput. Biol. Biomed.*, 2012, pp. 369–376.

[18] 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," *Nat. Biotechnol.*, vol. 33, no. 3, pp. 290–295, 2015.

[19] E. Bernard, L. Jacob, J. Mairal, and J.-P. Vert, "Efficient RNA isoform identification and quantification from RNA-Seq data with network flows," *Bioinformatics*, vol. 30, no. 17, pp. 2447–2455, 2014.

[20] J. J. Li, C.-R. Jiang, J. B. Brown, H. Huang, and P. J. Bickel, "Sparse linear modeling of next-generation mRNA sequencing (RNA-Seq) data for Isoform discovery and abundance estimation," *Proc. Nat. Acad. Sci. USA*, vol. 108, no. 50, pp. 19 867–19 872, 2011.

[21] A. I. Tomescu, A. Kuosmanen, R. Rizzi, and V. Mäkinen, "A novel combinatorial method for estimating transcript expression with RNA-Seq: bounding the number of paths," in *Proc. 13th Int. Workshop Algorithms Bioinf.*, 2013, pp. 85–98.

[22] A. I. Tomescu, A. Kuosmanen, R. Rizzi, and V. Mäkinen, "A novel min-cost flow method for estimating transcript expression with RNA-Seq," *BMC Bioinf.*, vol. 14, no. 5, pp. 1–10, 2013.

[23] Y.-Y. Lin *et al.*, "CLIIQ: Accurate comparative detection and quantification of expressed Isoforms in a population," in *Proc. 12th Int. Workshop Algorithms Bioinf.*, 2012, pp. 178–189.

[24] T. Gatter and P. F. Stadler, "Ryūtō: Network-flow based transcriptome reconstruction," *BMC Bioinf.*, vol. 20, no. 1, pp. 1–14, 2019.

[25] S. Mao, L. Pachter, D. Tse, and S. Kannan, "RefShannon: A Genome-guided transcriptome assembler using sparse flow decomposition," *PLoS One*, vol. 15, no. 6, 2020, Art. no. e0232946.

[26] L. Williams, A. I. Tomescu, and B. Mumey, "Flow decomposition with subpath constraints," in *Proc. 21st Int. Workshop Algorithms Bioinf.*, 2021, pp. 16:1–16:15.

[27] N. Eriksson *et al.*, "Viral population estimation using pyrosequencing," *PLoS Comput. Biol.*, vol. 4, no. 5, 2008, Art. no. e1000074.

[28] O. Zagordi, A. Bhattacharya, N. Eriksson, and N. Beerenwinkel, "ShoRAH: Estimating the genetic diversity of a mixed sample from next-generation sequencing data," *BMC Bioinf.*, vol. 12, no. 1, pp. 1–5, 2011.

[29] J. A. Baaijens, L. Stougie, and A. Schönhuth, "Strain-aware assembly of genomes from mixed samples using flow variation graphs," in *Proc. 24th Annu. Conf. Res. Comput. Mol. Biol.*, 2020, pp. 221–222.

[30] J. A. Baaijens, B. Van der Roest , J. Köster, L. Stougie, and A. Schönhuth, "Full-length de novo viral quasispecies assembly through variation graph construction," *Bioinformatics*, vol. 35, no. 24, pp. 5086–5094, 2019.

[31] R. P. Dilworth, "A decomposition theorem for partially ordered sets," *Ann. Math.*, vol. 51, no. 1, pp. 161–166, 1950. [Online]. Available: http://www.jstor.org/stable/1969503

[32] D. R. Fulkerson, "Note on Dilworthâs decomposition theorem for partially ordered sets," *Proc. Amer. Math. Soc.*, vol. 7, no. 4, pp. 701–702, 1956.

[33] J. Bang-Jensen and G. Z. Gutin, *Digraphs: Theory, Algorithms and Applications*. Berlin, Germany: Springer, 2008.

[34] V. Mäkinen, A. I. Tomescu, A. Kuosmanen, T. Paavilainen, T. Gagie, and R. Chikhi, "Sparse dynamic programming on DAGs with small width," *ACM Trans. Algorithms*, vol. 15, no. 2, pp. 1–21, 2019.

[35] M. Cáceres, M. Cairo, B. Mumey, R. Rizzi, and A. I. Tomescu, "Sparsifying, shrinking and splicing for minimum path cover in parameterized linear time," 2021, *arXiv:2107.05717*.

[36] A. Srivastava *et al.*, "Alignment and mapping methodology influence transcript abundance estimation," *Genome Biol.*, vol. 21, no. 1, pp. 1–29, 2020.

[37] J. D. Kececioglu and E. W. Myers, "Combinatorial algorithms for DNA sequence assembly," *Algorithmica*, vol. 13, no. 1, pp. 7–51, 1995.

[38] D. Li, C.-M. Liu, R. Luo, K. Sadakane, and T.-W. Lam, "MEGAHIT: An ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph," *Bioinformatics*, vol. 31, no. 10, pp. 1674–1676, 2015.

[39] J. Ruan and H. Li, "Fast and accurate long-read assembly with wtdbg2," *Nat. Methods*, vol. 17, no. 2, pp. 155–158, 2020.

[40] R. Rizzi, A. I. Tomescu, and V. Mäkinen, "On the complexity of minimum path cover with subpath constraints for multi-assembly," *BMC Bioinf.*, vol. 15, no. 9, pp. 1–11, 2014.

[41] S. C. Ntafos and S. L. Hakimi, "On path cover problems in digraphs and applications to program testing," *IEEE Trans. Softw. Eng.*, vol. SE-5, no. 5, pp. 520–529, Sep. 1979.

[42] H. V. Jagadish, "A compression technique to materialize transitive closure," *ACM Trans. Database Syst.*, vol. 15, no. 4, pp. 558–598, 1990.

[43] W. Pijls and R. Potharst, "Another note on Dilworth's decomposition theorem," *J. Discrete Math.*, vol. 2013, 2013, Art. no. 692645.

[44] N. Cloonan *et al.*, "Stem cell transcriptome profiling via massive-scale mRNA sequencing," *Nat. Methods*, vol. 5, no. 7, pp. 613–619, 2008.

[45] A. Mortazavi, B. A. Williams, K. McCue, L. Schaeffer, and B. Wold, "Mapping and quantifying mammalian transcriptomes by RNA-Seq," *Nat. Methods*, vol. 5, no. 7, pp. 621–628, 2008.

[46] P. M. Kim *et al.*, "Analysis of copy number variants and segmental duplications in the human Genome: Evidence for a change in the process of formation in recent evolutionary history," *Genome Res.*, vol. 18, no. 12, pp. 1865–1874, 2008.

[47] N. López-Bigas, B. Audit, C. Ouzounis, G. Parra, and R. Guigó, "Are splicing mutations the most frequent cause of hereditary disease?," *FEBS Lett.*, vol. 579, no. 9, pp. 1900–1903, 2005.

[48] Z. Wang, M. Gerstein, and M. Snyder, "RNA-Seq: A revolutionary tool for transcriptomics," *Nat. Rev. Genet.*, vol. 10, no. 1, pp. 57–63, 2009.

[49] S. P. Shah *et al.*, "The clonal and mutational evolution spectrum of primary triple-negative breast cancers," *Nature*, vol. 486, no. 7403, pp. 395–399, 2012.

[50] Q. Pan, O. Shai, L. J. Lee, B. J. Frey, and B. J. Blencowe, "Deep surveying of alternative splicing complexity in the human transcriptome by high-throughput sequencing," *Nat. Genet.*, vol. 40, no. 12, pp. 1413–1415, 2008.

[51] A. Byrne, C. Cole, R. Volden, and C. Vollmers, "Realizing the potential of full-length transcriptome sequencing," *Philos. Trans. Roy. Soc. B*, vol. 374, no. 1786, 2019, Art. no. 20190097.

[52] S. L. Amarasinghe, S. Su, X. Dong, L. Zappia, M. E. Ritchie, and Q. Gouil, "Opportunities and challenges in long-read sequencing data analysis," *Genome Biol.*, vol. 21, no. 1, pp. 1–16, 2020.

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

[54] K. Sahlin and V. Mäkinen, "Accurate spliced alignment of long RNA sequencing reads," *Bioinformatics*, Jul. 2021, Art. no. btab540. [Online]. Available: https://doi.org/10.1093/bioinformatics/btab540

[55] S. Kovaka, A. V. Zimin, G. M. Pertea, R. Razaghi, S. L. Salzberg, and M. Pertea, "Transcriptome assembly from long-read RNA-Seq alignments with StringTie2," *Genome Biol.*, vol. 20, no. 1, pp. 1–13, 2019.

[56] M. Guttman *et al.*, "Ab initio reconstruction of cell type–specific transcriptomes in mouse reveals the conserved multi-exonic structure of lincRNAs," *Nat. Biotechnol.*, vol. 28, no. 5, pp. 503–510, 2010.

[57] A. M. Mezlini *et al.*, "iReckon: Simultaneous isoform discovery and abundance estimation from RNA-Seq data," *Genome Res.*, vol. 23, no. 3, pp. 519–529, 2013.

[58] Z. Xia, J. Wen, C.-C. Chang, and X. Zhou, "NSMAP: A method for spliced isoforms identification and quantification from RNA-Seq," *BMC Bioinf.*, vol. 12, no. 1, pp. 1–13, 2011.

[59] D. Hiller and W. H. Wong, "Simultaneous isoform discovery and quantification from RNA-Seq," *Statist. Biosci.*, vol. 5, no. 1, pp. 100–118, 2013.

[60] L. Maretty, J. A. Sibbesen, and A. Krogh, "Bayesian transcriptome assembly," *Genome Biol.*, vol. 15, no. 10, pp. 1–11, 2014.

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

[62] T. Steijger *et al.*, "Assessment of transcript reconstruction methods for RNA-seq," *Nat. Methods*, vol. 10, no. 12, pp. 1177–1184, 2013.

[63] D. Deblasio, K. Kim, and C. Kingsford, "More accurate transcript assembly via parameter advising," *J. Comput. Biol.*, vol. 27, no. 8, pp. 1181–1189, 2020.

[64] C. Ma, H. Zheng, and C. Kingsford, "Finding ranges of optimal transcript expression quantification in cases of non-identifiability," *Biorxiv*, 2020, doi: 10.1101/2019.12.13.875625.

[65] C. Ma, H. Zheng, and C. Kingsford, "Exact transcript quantification over splice graphs," *Algorithms Mol. Biol.*, vol. 16, no. 1, pp. 1–15, 2021.

[66] A. D. Yates *et al.*, "Ensembl 2020, " *Nucleic Acids Res.*, vol. 48, no. D1, pp. D682–D688, 2020.

[67] V. Mäkinen, D. Belazzougui, F. Cunial, and A. I. Tomescu, *Genome-Scale Algorithm Design*. Cambridge, U.K.: Cambridge Univ. Press, 2015.

[68] E. A. Dinic, "Algorithm for solution of a problem of maximum flow in networks with power estimation," *Sov. Math. Doklady*, vol. 11, pp. 1277–1280, 1970.

[69] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, no. 2, pp. 248–264, 1972.

[70] B. Dezső, A. Jüttner, and P. Kovács, "LEMON–An open source C++ graph template library," *Electron. Notes Theor. Comput. Sci.*, vol. 264, no. 5, pp. 23–45, 2011.

[71] K. Damian, B. Comm, and M. Garret, "The minimum cost flow problem and the network simplex method," PhD Dissertation, Univ. College Dublin, Dublin, Ireland, 1991.

[72] G. B. Dantzig, *Linear Programming and Extensions*, vol. 48. Princeton, NJ, USA: Princeton Univ. Press, 1998.

[73] S. L. Salzberg *et al.*, "GAGE: A critical evaluation of genome assemblies and assembly algorithms," *Genome Res.*, vol. 22, no. 3, pp. 557–567, 2012.

[74] C. Jain, S. Misra, H. Zhang, A. Dilthey, and S. Aluru, "Accelerating sequence alignment to graphs," in *Proc. Int. Parallel Distrib. Process. Symp.*, 2019, pp. 451–461.

[75] M. Cáceres, M. Cairo, B. Mumey, R. Rizzi, and A. I. Tomescu, "A linear-time parameterized algorithm for computing the width of a DAG," in *Proc. Int. Workshop Graph-Theor. Concepts Comput. Sci.*, 2021, pp. 257–269.

Manuel Cáceres is currently working toward the doctoral degree in computer science at the University of Helsinki, Helsinki, Finland. His main research interests include graph algorithms, algorithmic bioinformatics, and compressed data structures.

Brendan Mumey received the PhD degree in computer science and engineering from the University of Washington, Seattle, Washington, in 1997 and joined the faculty with 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 with the University of Helsinki. He served as the editor of the *ACM SIGACT News*, a quarterly periodical for the theoretical computer science community, from 2008 to 2015.

Edin Husić is currently working toward the PhD degree in mathematics at the London School of Economics and Political Science, London, U.K. His main research interests include algorithmic game theory, combinatorial optimization, and graph theory.

Romeo Rizzi received the laurea degree in electronic engineering from the Politecnico di Milano, Milan, Italy, in 1991, and the PhD degree in computational mathematics and informatics from the University of Padova, Padua, Italy, in 1997. Afterwards, he held postdoc and other temporary positions at research centers like CWI (Amsterdam, Holland), BRICS (Aarhus, Denmark) and FBK (Trento, Italy). In 2001, he became assistant professor by the University of Trento. In 2005, he became associate professor by the University of Udine. Currently, he is full professor in operations research with the University of Verona, Italy. His main interests include combinatorial optimization and algorithms. He is an area editor of 4OR and acts as a reviewer for the American Mathematical Society. His papers cover the areas of discrete mathematics, combinatorics, algorithms, bioinformatics, and artificial intelligence. Since 2004, he has intensively acted as a trainer of the Italian team for the iOi.

Massimo Cairo received the PhD degree in mathematics from the University of Verona, Verona, Italy, in 2019, working on algorithmic problems related to graphs and temporal planning. During and after his PhD studies, he collaborated on multiple occasions with the Algorithmic Bioinformatics Research Group, University or Helsinki.

Kristoffer Sahlin received the PhD degree in computer science from the KTH Royal institute of Technology, Stockholm, Sweden, in 2015 and has worked as a postdoctoral researcher with Pennsylvania State University and the University of Helsinki. He is an assistant professor with the Department of Mathematics, Stockholm University and SciLifeLab fellow with the National Center for Molecular Biosciences, Science for Life Laboratory.

Alexandru I. Tomescu received the PhD degree in computer science from the University of Udine, Udine, Italy, in 2012. Between 2012 and 2020, he worked with 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 with 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.