Safe Sequences via Dominators in DAGs for Path-Covering Problems

Francisco Sena[1]
Department of Computer Science, University of Helsinki, Finland

Romeo Rizzi
Department of Computer Science, University of Verona, Italy

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

Abstract

A *path-covering* problem on a directed acyclic graph (DAG) requires finding a set of source-to-sink paths that cover all the nodes, all the arcs, or subsets thereof, and additionally they are optimal with respect to some function. In this paper we study *safe sequences* of nodes or arcs, namely sequences that appear in some path of every path cover of a DAG.

We show that safe sequences admit a simple characterization via cutnodes. Moreover, we establish a connection between maximal safe sequences and leaf-to-root paths in the source- and sink-dominator trees of the DAG, which may be of independent interest in the extensive literature on dominators. With dominator trees, safe sequences admit an $O(n)$-size representation and a linear-time output-sensitive enumeration algorithm running in time $O(m + o)$, where n and m are the number of nodes and arcs, respectively, and o is the total length of the maximal safe sequences.

We then apply maximal safe sequences to simplify Integer Linear Programs (ILPs) for two path-covering problems, LeastSquares and MinPathError, which are at the core of RNA transcript assembly problems from bioinformatics. On various datasets, maximal safe sequences can be computed in under 0.1 seconds per graph, on average, and ILP solvers whose search space is reduced in this manner exhibit significant speed-ups. For example on graphs with a large width, average speed-ups are in the range $50 - 250\times$ for MinPathError and in the range $80 - 350\times$ for LeastSquares. Optimizing ILPs using safe sequences can thus become a fast building block of practical RNA transcript assembly tools, and more generally, of path-covering problems.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Integer programming; Applied computing → Sequencing and genotyping technologies

Keywords and phrases directed acyclic graph, path cover, dominator tree, integer linear programming, least squares, minimum path error

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.55

Related Version *Full Version*: `https://arxiv.org/abs/2411.03871`

Supplementary Material *Software (Source Code)*: `https://github.com/algbio/safe-sequences`
archived at `swh:1:dir:0e6e4d5e013053c8ba853d335c750cff701e09fa`

Funding This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 851093, SAFEBIO), and from the Research Council of Finland grants No. 346968, 358744. Co-funded by the European Union (ERC, SCALEBIO, 101169716). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Acknowledgements We are extremely grateful to Andrey Prjibelski for producing the IsoQuant graphs and to Manuel Cáceres for fruitful discussions on safe sequences.

[1] Corresponding author

1 Introduction

Data reduction, namely simplifying the input to a problem while maintaining equivalent solutions, has proved extremely successful in solving hard problems in both theory and practice [26]. While the most well-known approach for data reduction is *kernelization* [37, 18], there are also other ways to pre-process the input in order to obtain practical algorithms.

For example, for graph problems where a solution is a set of vertices with certain properties, Bumpus et al. [8] and Jansen and Verhaegh [29] considered the notion of *c-essential vertex* as one belonging to *all* c-approximate solutions to the problem. They showed that for some vertex-subset finding problems (e.g. Vertex Cover) and some values of c, there exist algorithms with an exponential dependency only in the number of *non-c-essential vertices*.

In this paper we pursue a similar data reduction approach, via the notion of *safe partial solution* [52], defined as one "appearing" in all solutions to a problem. For solutions that are sets of paths satisfying certain properties, a *safe path* can be defined as one appearing as a subpath of some path of any solution. Intuitively, safety corresponds to 1-essentiality.

For an NP-hard problem, it is usually intractable also to compute all its safe partial solutions (see e.g. [15]). As such, in this paper we use the following property about safety. Let $\mathcal{S}$ be the (unknown) set of solutions to a problem, and let $\mathcal{S}' \supseteq \mathcal{S}$ be a superset of $\mathcal{S}$ whose safe partial solutions can be computed efficiently. Then, if a partial solution appears in all solutions in $\mathcal{S}'$, it also appears in all solutions in $\mathcal{S}$.

This property was used by Grigorjew et al. [25] when applying safety as a pre-processing step for the Minimum Flow Decomposition (MFD) problem. The MFD problem asks to decompose the flow in a directed acyclic graph (DAG) into a minimum number of weighted source-to-sink paths [4, 53]. As solution superset $\mathcal{S}'$, they considered flow decompositions with any number of paths, whose safe paths can be computed in polynomial time [30, 31, 2]. Instead of simplifying the input graph, they used the safe paths to simplify an integer linear program (ILP) for the MFD problem from [17], thus reducing the search space of the ILP solver. This resulted in speed-ups of up to 200 times on the hardest instances.

In this paper we focus on problems whose input consists of an *s-t DAG*[2] G where the solution superset $\mathcal{S}'$ contains all the sets of paths in G from s to t that together cover all the nodes or arcs of G (or given subsets thereof). We will refer to these objects simply as *path covers*. Note that a flow decomposition is also a path cover of the arcs carrying flow, but path covers generalize many other problems, especially those modeling practical problems from bioinformatics, see Section 1.2. Next, we present the contributions of the paper and further contextualization.

1.1 Optimal enumeration of maximal safe sequences via dominators

Safe sequences as a generalization of safe paths. The notion of a safe *path* has been extensively used to capture *contiguous* safe partial solutions (see e.g. [52, 48, 15, 40, 45, 31, 11]). However, for our purposes contiguity is not necessary, but only the fact that the nodes/arcs in a safe partial solution appear in the same order in some path of every solution. For example, it may be that the only way to reach a node u is via a node v, with some complex subgraph between them. The sequence u, v thus appears in some path of every path cover of the nodes.

[2] By an *s-t directed graph* we mean one with a unique source s and unique sink t such that all nodes are reachable from s and all nodes reach t; if the graph is acyclic, then it is an *s-t DAG*. Path-covering problems on DAGs typically require that paths either start in a source of the graph, or more generally, in a given set of starting nodes, and analogously end in a sink, or in a given set of terminal nodes. However, these start/end nodes can naturally be connected to s and t, and thus we can assume, without loss of generality, that our DAGs are s-t DAGs.

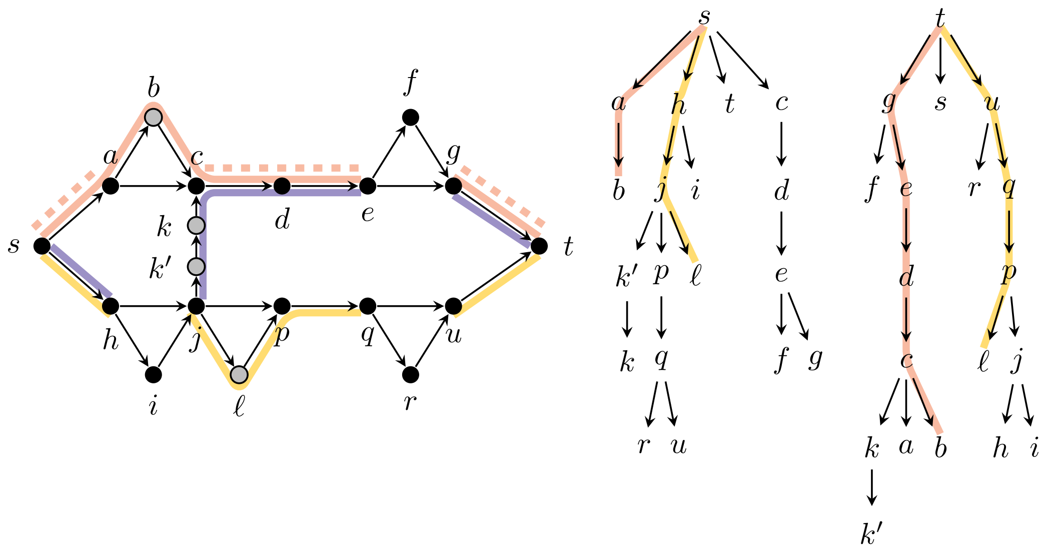

Figure 1 An s-t DAG and its s- and t-dominator trees. Three maximal safe sequences with respect to path covers of the nodes are shown in solid orange, violet, and yellow. Nodes b and ℓ are leaves in both dominator trees, and thus their extension into the orange and yellow sequences, resp., are maximally safe. (The maximal safe sequences obtained by extending nodes f, i and r are not shown.) Note that $k'k$ is a unitary path and that $\mathsf{extension}(k') = \mathsf{extension}(k)$. The dashed orange safe sequence is not maximal and can be obtained by extending a.

We thus define a *safe sequence* with respect to the path covers of a DAG G, either of the nodes or of the arcs, analogously to that of a safe path (see Figure 1 for an example). This notion is inspired by the work of Ma, Zheng and Kingsford [57], who proposed a problem (AND-Quant) and a max-flow-based algorithm deciding when a sequence of arcs appears in some path in every flow decomposition (with any number of paths) of a flow in a DAG. Here, we are particularly interested in characterizing but also in computing *maximal* safe sequences, i.e., safe sequences that are not a proper subsequence of another safe sequence.

Characterization of safe sequences. Assume that the input DAG has n nodes and m arcs. We show that safe sequences admit a simple characterization in terms of cutnodes. A *u-v cutnode* is a node such that all paths from u to v (*u-v paths*) pass through it. We present our results for path covers of the nodes and refer to the extended version for the results for path covers of the arcs. Omitted proofs can be found in the extended version.

▶ **Theorem 1** (Characterization of safe sequences). *Let $G = (N, A)$ be an s-t DAG and X be a sequence of nodes. The following statements are equivalent:*

1. *X is safe for path covers.*
2. *There is a node $u \in N$ such that every s-t path covering u contains X.*
3. *There is a node $v \in N$ such that X is contained in the sequence obtained as the concatenation of the sequences of s-v and v-t cutnodes.*

While outputting the sequences of s and t cutnodes for every node v in the graph can be done in time $O(m)$ per node, different nodes can thus be extended to the same maximal safe sequence, and some extensions may not even be maximal (see Figure 1). The challenge is to characterize those nodes identifying all and only maximal safe sequences, without duplicates.

Linear-time output-sensitive enumeration via dominators. We show that we can identify such nodes in an elegant manner using *dominators*, which is a well-studied concept related to cutnodes (see e.g. the survey [23] for an in-depth contextualization of dominators). In particular, the *s-dominator tree* of an s-t DAG (N, A) has node set N, and u is the parent of v iff all paths from s to v pass through u, and moreover there is no node reachable from u (except u itself) whose removal makes v unreachable from u.

The key idea behind our results is to work *also* with the t-dominator tree of G, which is defined in an analogous manner, but considering reachabilities *to t*. By analyzing the interplay between the two trees, we show that the nodes solving this problem (Theorem 9) are those that are *leaves in both* the s- and t-dominator trees, only after overcoming some technicalities arising from unitary paths (see Figure 1, Lemma 4 and Lemma 8). Since dominator tree leaves admit a characterization in terms of in- or out-neighborhoods on DAGs (Lemma 5), this leads to an $O(mn)$-time algorithm outputting all and only maximal safe sequences without duplicates, independently of dominator trees:

▶ **Theorem 2.** *Given an s-t DAG G with n nodes and m arcs, we can compute all and only the maximal safe sequences of G in $O(nm)$-time, without constructing dominator trees.*

To improve this time bound we resort to the fact that dominator trees admit an $O(m+n)$ time construction [6, 7, 20], although so far some of these algorithms do not seem to be practical. See [24, 22, 14] for experimental studies on different dominator tree algorithms. Regardless, any dominator algorithm serves our purposes and its effect on the runtime of our algorithm is clear in the proof of the theorem below.

▶ **Theorem 3** (Enumeration of maximal safe sequences). *Let G be an s-t DAG with n nodes and m arcs. The following hold:*

1. *There is an $O(m + o)$ time algorithm outputting the set of all maximal safe sequences with no duplicates, where o denotes the total length of all the maximal safe sequences.*
2. *There is an $O(n)$-size representation of all the maximal safe sequences of G that can be built in $O(m + n)$ time.*

An analogous result to Theorem 3 can be obtained for path covers where only a given subset $C \subseteq N$ of nodes has to be covered. To achieve that, we have to analyze the dominator trees induced by the dominance relation restricted to C. Moreover, analogous results can be obtained also for path covers of the arcs. The details can be found in the extended version.

1.2 Scaling path-covering ILPs for bioinformatics problems

The motivation of our work comes from bioinformatics, where many NP-hard graph problems related to sequencing data require finding a set of paths in a DAG satisfying certain properties. For example, in the *RNA transcript assembly* problem, we need to recover the RNA transcripts produced in various abundances by a gene, given only smaller fragments (i.e. RNA-seq reads) sequenced from the RNA transcripts. This problem is usually solved by first building an arc-weighted DAG from the fragments, i.e., a *splice graph*. The RNA transcripts correspond to a set of source-to-sink weighted paths in the DAG that "best explain" the arc-weights, under various definitions of optimality, see e.g. [36, 35, 56, 46, 55, 47, 50, 19, 17, 15].

In our work we consider two path-covering problems used for RNA transcript assembly, in which we use safe sequences to fix some variables of the corresponding ILPs. We assume that in every arc uv of the input graph there is an associated positive weight $w(uv)$. We now briefly describe the ILPs.

We selected a classical least-squares model (which we call LeastSquares) that is at the core of several RNA assembly tools e.g. [34, 12, 21, 33, 38, 51]. This minimizes the sum of squared differences between the weight of an arc and the weights of the solution paths passing through the arc. The ILP formulation requires no additional constraints, and the ILP objective function is:[3]

$$\min \sum_{uv \in A} \left(w(uv) - \sum_{i=1}^{k} x_{uvi} w_i \right)^2 . \tag{1}$$

While this model is popular, it has quadratic terms in its objective function, which makes it harder to solve. Therefore, as second model (which we call MinPathError), we chose one that was recently introduced in [16]. Then, for k paths to be a solution to the ILP, we require that the absolute difference between the weight of an arc and the weights of the solution paths passing through the arc must be below the sum of the errors (or slacks) of the paths passing through the arc. Formally, we add the constraint:[4]

$$\left| w(uv) - \sum_{i=1}^{k} x_{uvi} w_i \right| \leq \sum_{i=1}^{k} x_{uvi} \rho_i, \quad \forall uv \in A, \tag{2}$$

with objective function: $\min \sum_{i=1}^{k} \rho_i$. By definition, the solution paths must cover every arc uv, since otherwise the above condition becomes $w(uv) \leq 0$, which contradicts the assumption that w is positive everywhere. See [17] for further details.

On various datasets, maximal safe sequences can be computed in under 0.1 seconds per graph, on average, and such safety-optimized ILPs show significant speed-ups over the original ones. More specifically, on the harder instances of graphs with a large width, average speed-ups are in the range $50 - 250\times$ for MinPathError and in the range $80 - 350\times$ for LeastSquares. As such, our optimizations can become a scalable building block of practical RNA transcript assembly tools, but also of general path-covering problems.

2 Notation and preliminaries

Graphs. In this paper we consider directed graphs, especially directed acyclic graphs (DAGs). For a DAG $G = (N, A)$ with node set N and arc set A, we let $n = |N|$ and $m = |A|$. We denote an arc from u to v as uv and a path p through nodes $v_1, \dots, v_k$ as $p = v_1 \dots v_k$. Two paths are considered distinct if they differ in at least one node. The set of *in-neighbors* (*out-neighbors*) of a node v is denoted N_v^- (N_v^+), and its *indegree* (*outdegree*) as d_v^- (d_v^+). For nodes u, v we say that there is a *u-v path* if there is a path whose first node is u and last node is v (we also say that u *reaches* v). An *antichain* is a set of pairwise unreachable nodes. The size of the largest antichain in a DAG is called *width*. Analogously, for arcs uv, xy we say that uv reaches xy if v reaches x. An *arc-antichain* is a set of pairwise unreachable arcs, and the *arc-width* of a DAG is the size of the largest arc-antichain of the graph.

For $X \subseteq N$ we denote a *sequence* X through nodes $u_1, \dots, u_\ell$ as $X = u_1, \dots, u_\ell$ if there is a u_i-u_{i+1} path for all $1 \leq i \leq \ell - 1$; in particular, a path is a sequence. We say that X *covers* or *contains* a node v if $v = u_i$ for some $i \in \{1, \dots, \ell\}$, and we write $u \in X$. If Y is a sequence and each node of X is contained in Y we say that X is a *subsequence* of Y, or that X is *contained* in Y. Since G is a DAG, the nodes of X appear in the same order as in Y.

[3] Note that the terms $x_{uvi} w_i$ are not linear, but can be linearized using a simple technique, see [17, 16].
[4] Note again that the terms $x_{uvi} w_i$ and $x_{uvi} \rho_i$ are not linear, but can be linearized using a simple technique, see [16].

Let $X' = v_1, v_2, \dots, v_{\ell'}$ be a sequence such that there is a path from the last node u_ℓ of X to v_1. The *concatenation* of X and X' is simply the sequence XX' obtained as X followed by X'. If $u_\ell = v_1$, then the repeated occurrence of v_1 is removed, i.e. $XX' := u_1, \dots, u_\ell, v_2, \dots, v_{\ell'}$. We adopt the convention that lowercase letters u, v, w, x, y, z denote nodes and uppercase letters X, Y denote sequences of nodes.

Safety. We define a *path cover* of G to be a set P of s-t paths such that $\forall u \in N\ \exists p \in P : u \in p$; we also say *cover* instead of "s-t path cover of G" when the graph G is clear from the context. Two path covers are distinct if they differ in at least one path. We assume that every node of G lies in some s-t path. We say that a sequence X is *safe* for G if in every path cover P of G it holds that X is a subsequence of some path in P. If X is not safe we say that it is *unsafe*. A sequence X is *maximally safe* if it is not a subsequence of any other safe sequence. We assume that there is always an s-t path containing a given sequence of nodes, otherwise that sequence is vacuously unsafe. Note that imposing the collection of s-t paths of a cover to be minimal with respect to inclusion results in an equivalent definition of safety: any sequence that is safe for path covers is also safe for minimal path covers since every minimal path cover is a path cover; but also, if X is safe for minimal path covers then it is also safe for general path covers, as otherwise we could have a path cover P avoiding X from which arbitrary s-t paths can be removed until P becomes minimal.

Graph compression. We define a *unitary path* as a path where each node has indegree equal to one but the first one, and each node has outdegree equal to one but the last one. For example, in Figure 1, the paths *de* and *pq* are unitary, the path *cde* is maximally unitary. With respect to G, we define the *compressed graph of* G as the graph where every maximal unitary path of G is contracted into a single node. It is a folklore result that compressing a graph can be done in linear time (see e.g. [13, 52]). We state this fact in the next lemma together with an additional property stating that compressed graphs cannot have arcs uv with v the unique out-neighbor of u and u the unique in-neighbor of v, since uv would then be a unitary path.

▶ **Lemma 4.** *Let G be a directed graph with n nodes and m arcs. Then the following hold:*
1. *We can compute the compressed graph of G in $O(n + m)$ time.*
2. *In a compressed graph there is no arc uv such that $N^+_u = \{v\}$ and $N^-_v = \{u\}$.*

The compression of unitary paths preserves safety: every s-t path in G traversing the unitary path traverses the corresponding node in the compressed graph of G, and every s-t path traversing a node in the compressed graph of G necessarily traverses the unitary path it corresponds to in G. This idea appeared in the context of walk-finding problems in [11].

Cutnodes and dominators. In the dominator literature, the directed graphs on which dominators are computed have a single start node s from which every node is reachable. In some cases, the graphs have instead or additionally a unique sink t which is reached by every node, in which case the dominance relation is referred to as post-dominance (see e.g., [5, 27]). A prominent application of dominators is in the context of program analysis, where s corresponds to the entry point of the program and t to its exit point. Our graphs always have unique abstract nodes s and t, hence, diverging from standard notation, we parameterize the domination relation by s and t. We borrow standard concepts and notation from the literature of dominators, which we describe next.

Let G be an s-t directed graph. A node u is said to *s-dominate* a node v if every s-v path contains u, hence u is an s-v cutnode if and only if u s-dominates v. Every node s- and t-dominates itself. We also say that v *t-dominates* u if every u-t path contains v, which is equivalent to v being a u-t cutnode. We say that u *strictly s-dominates* v if u s-dominates v and $u \neq v$. For $v \neq s$, the *immediate s-dominator* of v is the strict s-dominator of v that is dominated by all the nodes that strictly dominate v, in which case we write $idom_s(v) = u$. The domination relation on G with respect to s can be represented as a rooted tree at s with the same nodes as G, called the *s-dominator tree* of G. In this tree, the parent of every node is its immediate s-dominator, every node is s-dominated by all its ancestors and s-dominates all its descendants and itself. A node is said to be an *s-dominator* if it strictly s-dominates some node. The same terminology will be used for the t-domination relation. We also define a function $dom : N \times \mathbb{N}^+ \to N$ on rooted trees as follows. Let $dom_s(v,k) := idom_s(v)$ when $k = 1$ and $dom_s(v,k) := dom_s(dom_s(v,1), k-1)$ when $k > 1$. If k is larger than the number of *strict* dominators of a given node, then it defaults to s (resp. t) in the s-dominator tree (resp. t-dominator tree). Essentially, dom gives the k-th ancestor of a node in a rooted tree, if it is well defined, otherwise it returns the root of the tree. For example, in Figure 1, the immediate s-dominator of b is a ($dom_s(b,1) = a$) and $dom_t(q,3) = t$. Node c is a strict t-dominator of the nodes $\{a, b, k, k'\}$. See [23] for an in-depth presentation of dominators.

In DAGs, dominators admit a characterization, which we state in Lemma 5 below. This was first presented by Ochranová [41], but with an incomplete proof. This characterization was also discussed by Alstrup et al. [6, Sec. 5.1]. To the best of our knowledge, we did not find a complete proof of this characterization in the literature, and hence one can be found in the extended version.

▶ **Lemma 5** (Characterization of node-dominators in DAGs). *Let $G = (N, A)$ be an s-t DAG. A node $u \in N$ is an s-dominator (resp. t-dominator) if and only if there is a node v such that $N_v^- = \{u\}$ (resp. $N_v^+ = \{u\}$).*

3 Safe sequences via dominators

3.1 Characterization of safe sequences

We start by proving Theorem 1.

Proof of Theorem 1. ($1 \Rightarrow 2$) If for every node $u \in N$ there is an s-t path covering u that avoids X, then we can build a path cover by considering for every node u of G an s-t path avoiding X that covers u. This contradicts the safety of X.

($2 \Rightarrow 3$) Since every s-t path covering u contains the sequence of s-u cutnodes followed by the sequence of u-t cutnodes and X is contained in every such path, the implication follows by taking $u = v$.

($3 \Rightarrow 1$) Since X is a subsequence of a sequence containing every s-v and v-t cutnode and v must be covered in every path cover by some s-t path, any of which contains the s- and t-cutnodes of v, it follows that X is safe. ◀

We will heavily use the type of sequence obtained as in point 3 of Theorem 1. As such, we define the *extension* of a node v, $\mathsf{extension}(v)$, as the sequence obtained from the concatenation of the sequence of s-v cutnodes with the sequence of v-t cutnodes. By Theorem 1 every maximal safe sequence is the extension of some node, and so in a DAG with n nodes and m arcs there are at most n maximal safe sequences. Moreover, for any node $v \in N$, $\mathsf{extension}(v)$ can be computed in time $O(m+n)$ by computing the s-v cutnodes and v-t cutnodes with e.g. the simple algorithm of Cairo et al. [9]. Thus, Theorem 1 implies that we can compute in time $O(nm)$ a set of safe sequences containing every maximal safe sequence of G, by reporting $\mathsf{extension}(v)$ for every node v. However, the reported sequences may not be maximal and the same sequence may be reported multiple times.

Because of the correspondence between cutnodes and dominators explained in Section 2, the extension of a node can also be expressed as suitable paths in dominator trees. We will use this fact in the rest of the paper.

▶ Remark 6. In an s-t DAG G, $\mathsf{extension}(v)$ is the same as the path from s to v in the s-dominator tree of G concatenated with the path from v to t in the t-dominator tree of G.

3.2 Properties of s- and t-dominator trees

We start with a lemma showing an interplay between s-domination and t-domination on general s-t directed graphs.

▶ **Lemma 7.** *Let $G = (N, A)$ be an s-t directed graph, let $u, v \in N$ be nodes, and let $k \in \mathbb{N}^+$. If u is the k-th ancestor of v in the s-dominator tree, then $dom_t(u, k)$ t-dominates v, and v is not a t-dominator of u unless $v = dom_t(u, k)$.*

Proof. Note that $u \neq t$ since u is an s-dominator. Let $w = dom_t(u, k)$ and let $y_1, \ldots, y_{k-1}$ be the nodes between u and v in the s-dominator tree. We can assume $w \neq v$.

Observe that $w \neq y_i$ for all $i \in \{1, \ldots, k-1\}$. Suppose otherwise. Then $w \neq t$ because t is not an s-dominator. Hence, by definition of dom_t, there are $k-1$ nodes $x_1, \ldots, x_{k-1}$ between w and u in the t-dominator tree. Some x_j is not between u and w in the s-dominator tree, and hence there is a u-w path avoiding x_j. Further, since x_j is strictly t-dominated by w there is a w-t path avoiding x_j. Concatenating these paths at w results in a u-t path avoiding x_j, contradicting the fact that x_j t-dominates u.

Suppose now for a contradiction that v is not strictly t-dominated by w. From the point above, there is a u-v path avoiding w. This path concatenated with a v-t path avoiding w gives a u-t path also avoiding w, contradicting the fact that w t-dominates u.

It remains to show that v cannot t-dominate u. Suppose otherwise. Then we have $v = x_i$ for some $i \in \{1, \ldots, k-1\}$, since v is strictly t-dominated by w and t-dominates u. Thus, some y_j is not between v and u in the t-dominator tree, implying that there is a u-v path avoiding y_j. Since y_j is s-dominated by u, there is an s-u path avoiding y_j, which concatenated with the previous path gives an s-v path also avoiding y_j, contradicting the fact that y_j s-dominates v. ◀

Lemma 7 can be interpreted for the special case of $k = 1$ as follows: if u immediately s-dominates v then the immediate t-dominator of u t-dominates v (see Figure 2). Another consequence of this result is that if u is an s-dominator, then at most one node immediately s-dominated by u is in the path from t to u in the dominator tree of t, which must be the immediate t-dominator of u.

Observe now in Figure 1 that c is the immediate s-dominator only of d and that d is the immediate t-dominator only of c, and likewise for d and e, and so $\mathsf{extension}(c) = \mathsf{extension}(d) = \mathsf{extension}(e)$. This redundancy and asymmetry introduced by unitary paths is undesirable in the characterization of maximal safe sequences. This discussion motivates the following result.

▶ **Lemma 8.** *Let $G = (N, A)$ be a compressed s-t DAG and let $u, v \in N$ be distinct nodes. If u immediately s-dominates v and v immediately t-dominates u then u immediately s-dominates some node different from v and v immediately t-dominates some node different from u.*

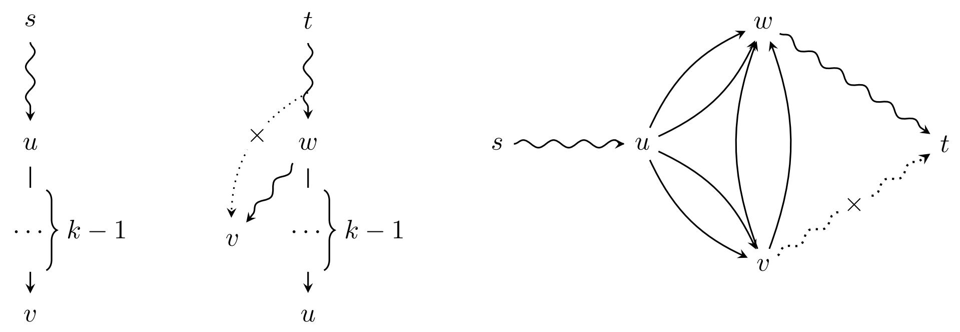

Figure 2 Illustration of the proof of Lemma 7. The two dominator trees discussed in the proof are shown on the left. On the right, an abstract graph for the special case of $k = 1$ is shown; node v cannot reach t without w, without contradicting the fact that $dom_t(u, 1) = w$.

Proof. Assume without loss of generality that u immediately s-dominates only node v.

We claim that $N_v^- = \{u\}$. Suppose otherwise. If N_v^- is empty then $v = s$, and so v is not a t-dominator, a contradiction. So there is an arc wv with $w \neq u$. Since u s-dominates v, it must also s-dominate every in-neighbour of v, namely w. But u immediately s-dominates only v, thus, v must s-dominate w, implying that there is a v-w path. Since $wv \in A$, G contains a cycle, a contradiction.

Since v is a t-dominator, by Lemma 5 there is a node whose unique out-neighbour is v. If this node is u, we contradict the fact that the graph is compressed by Lemma 4. Therefore, v must have at least two distinct in-neighbors, a contradiction. ◀

3.3 Characterization and enumeration of maximal safe sequences

Using Lemmas 7 and 8, we can give a precise characterization of maximal safe sequences.

▶ **Theorem 9** (Characterization of maximal safe sequences). *Let $G = (N, A)$ be a compressed s-t DAG and let $u \in N$ be a node. Then* extension(u) *is a maximal safe sequence if and only if u is a leaf in both the s- and t-dominator trees of G.*

Proof. ($\Leftarrow$) If u is a leaf in both dominator trees then extension(u) is maximal: no other node can be added to this sequence by the equivalence of safe sequences and paths in the dominator trees given by Remark 6 and Theorem 1.

($\Rightarrow$) Suppose that extension(u) is a maximal safe sequence but u is not a leaf in both dominator trees. Without loss of generality suppose that u is not a leaf in the s-dominator tree. Note that $u \neq t$ since u is an s-dominator. Let v be a node immediately s-dominated by u and let w denote the immediate t-dominator of u.

First we argue that there is a node z that is immediately s-dominated by u and that is strictly t-dominated by w. We consider two cases. If $v \neq w$ then we can take $z = v$, because by Lemma 7 for $k = 1$ we can conclude that w t-dominates v, and since $v \neq w$, it strictly t-dominates v. If $v = w$ then we can take $z = v'$, since by Lemma 8 there is a node $v' \neq v$ that is immediately s-dominated by u; applying Lemma 7 to u and v' with $k = 1$, we conclude that w t-dominates v, and since $w = v \neq v$, w strictly t-dominates v'.

Now we show that extension(z) properly contains extension(u). Consider extension(u), which by definition consists of the sequence of s-u cutnodes (call it Y_1) concatenated with the sequence of u-t cutnodes (call it Y_2). In extension(u), by definition, the double occurrence of u is removed, and thus let Y_2' be the sequence of w-t cutnodes. That is, extension(u) can be written as Y_1Y_2'. Consider also extension(z), which can be written analogously as X_1X_2'. Since u is the parent of z in the s-dominator tree, we have that Y_1 is *strictly* contained in X_1, and since z is a proper descendant of w in the t-dominator tree, we have that Y_2' is contained in X_2' (possibly $Y_2' = X_2'$). Therefore, extension(u) is a proper subsequence of extension(z), contradicting our assumption that extension(u) is a maximal safe sequence. ◀

The idea in the above theorem of targeting special nodes or arcs in order to characterize every maximal safe object is a recurring theme in safety problems. For instance, [31, 2] proposed the notion of *representative arc*, [48] proposed that of *core*, and [10] found a special class of walks, coined *macrotigs*, from where every maximal safe walk can be built. Each of these constructs was used in a different safety problem.

While the above characterization uses dominator trees, we can derive an equivalent characterization of dominator-tree leaves just in terms of in- and out-neighborhoods via Lemma 5, since a node is a leaf if and only if it is not a dominator. Computing the extension of every node satisfying such condition leads to a simple $O(mn)$ time algorithm outputting all and only maximal safe sequences, without needing to compute the dominator trees, from which Theorem 2 follows.

Since dominator trees can be built in $O(m + n)$ time (see, e.g., the algorithm by Alstrup et al. [6], Buchsbaum et al. [7]), or Fraczak et al. [20]), we can prove Theorem 3.

Proof of Theorem 3. Let G' be the compressed graph of G. Build the dominator trees of G' and compute all the leaves common to both trees. These three steps take $O(m + n)$ time.

1: For every node that is a leaf in both dominator trees, output extension(v). By Theorem 9 we find every maximal safe sequence during this process. Further, no duplicate safe sequence is computed since different leaves common to both trees have different extensions. Therefore the algorithm runs in time equal to the length of all the maximal safe sequences plus the time required to build two dominator trees over G', so $O(m + o)$ altogether.

2: With the equivalence given by Theorem 9, we have that every maximal safe sequence is the result of extension(v) for every node v of G' that is a leaf in both dominator trees. The sequence extension(v) is encoded in the dominator trees by Remark 6. ◀

4 Experiments

The experiments were performed on an AMD 32-core machine with 512GB RAM. The source code of our project is publicly available on Github[5]. All our algorithms are implemented in Python3, as are the scripts to produce the experimental results. To solve ILPs, we use Gurobi's Python API (version 11) under an academic license. For every graph, we run Gurobi with 4 threads and set up a timeout of 300 seconds in Gurobi's execution time.

Implementation. As described in Section 1.2, we study experimentally the potential of applying safe sequences in two arc-path covering problems, LeastSquares and MinPathError.

We did not use an external algorithm to compute arc-dominator trees because these are mostly tailored for node-dominators. Instead, we compute the immediate s-dominator and t-dominator of all arcs with an $O(m^2)$ time procedure via the algorithm of [9], appropriately handling the reversed graph of G for the immediate s-domination relation. The classical

[5] `https://github.com/algbio/safe-sequences`

algorithm of Aho and Ulman [3] to build dominator trees has the same running time as our procedure. We remark that even with a suboptimal algorithm to compute the dominator trees, our entire safety-preprocessing step takes less than 0.1 seconds on average.

With the dominator trees, we can compute all the maximal safe sequences with respect to arbitrary subsets $C \subseteq A$ of arcs to be covered, all in time proportional to the total length of all the maximal safe sequences, completely analogously to the discussion in Section 3.

The ILPs were optimized as in [25] by fixing x_{uvi} variables to 1 based on the maximal safe sequences of the input DAG, which reduces the search space of the linear program and hence speeds-up the solver. Let $X_1, \dots, X_t$ be sequences in the input DAG such that: (a) each X_i, $i \in \{1, \dots, t\}$, must appear in some solution path of the ILP, and (b) there exists no path in the DAG containing two distinct X_i and X_j. In the case that every X_i is a path, Grigorjew et al. [25] noticed that if the two mentioned properties hold, then $X_1, \dots, X_t$ must appear in t distinct paths of among the k solution paths of the ILP (see the three solid-colored sequences shown in Figure 1 which must appear in different paths of any path cover). As such, without loss of generality, one can set $x_{uvi} = 1$ for each arc uv of X_i, for all $i \in \{1, \dots, t\}$. When using sequences to set binary variables, we can proceed in a completely analogous manner. To have a large number of binary variables set to 1, we follow the approach of [25]. Namely, for every arc uv, we define its weight as the length of the longest safe sequence containing uv. Then we find a maximum-weight antichain of arcs in the DAG using the min-flow reduction of [39]. Then, for each a_i in the maximum-weight antichain, we consider a longest safe sequence X_i containing a_i to fix variables to 1. Note that a path covering two such sequences X_i and X_j would witness that a_i and a_j cannot be in an antichain. Finally, we set k to the arc-width of the graph and run Gurobi to solve the ILP.

Note that any solution to MinPathError is also a path cover, since each arc weight is positive, and thus the simplified ILP has equivalent optimal solutions. This may not be the case for LeastSquares, since some arcs may not be covered by any solution path. To handle such scenarios we propose using maximal safe sequences for path covers where only a subset C of nodes or arcs can be trusted to appear in any optimal solution. As an illustration of this approach, we perform experiments by choosing the subset C as those arcs whose weight is at least as large as the 25-th percentile of the weights of all arcs, since we can heuristically infer that the solution paths of LeastSquares cover the arcs of large enough weight.

Datasets. We experiment with seven datasets. The first four contain splice graphs with erroneous weights created directly from gene annotation by [16].[6] These were created starting from the well-known dataset of Shao and Kingsford [49], which was also used in several other studies benchmarking the speed of minimum flow decomposition solvers [17, 32, 54]. The original splice graphs were created in [49] from gene annotation from **Human**, **Mouse** and **Zebrafish**. This dataset also contains a set of graphs (**SRR020730**) from human gene annotation, but with flow values coming from a real sequencing sample from the Sequence Read Archive (SRR020730), quantified using the tool Salmon [42]. To mimic splice graphs constructed from real read abundances, [16] added noise to the flow value of each arc, according to a Poisson distribution. The last three datasets contain graphs constructed by a state-of-the-art tool for RNA transcript assembly, IsoQuant [44]. Mouse annotated transcript expression was simulated according to a Gamma distribution that resembles real expression profile [44]. From these expressed transcripts, PacBio reads were simulated using

[6] Available at `https://doi.org/10.5281/zenodo.10775004`

IsoSeqSim [1] and fed to IsoQuant for graph creation (we call this **Mouse PacBio**), and ONT reads were simulated using Trans-Nanosim [28], which was modified to perform a more realistic read truncation (see [44]), and fed to IsoQuant for graph creation (we call this **Mouse ONT**). For the latter one, there is also a simpler version where no read truncation was introduced (**Mouse ONT.tr**). These are available at [43].

Discussion. We group the graphs in each dataset by width (column "w"). For each width-interval we show different metrics, namely the number of graphs "#g", the average number of arcs "avg m" (and the maximum number of arcs in parenthesis), the time taken in seconds by the safety preprocessing step in "prep", the percentage of fixed arc-variables x_{uvi} of the ILP out of all mk variables in "vars", the number of solved instances by the ILP with and without safety in "#solved" (and in parenthesis the average time taken in seconds to solve those instances), and in the column "$\times$" we show the average speedup obtained by advising the linear program with safety. The speed-ups are computed by considering for each graph the ratio between the time taken by the original ILP (or 300 sec. if this timed-out) divided by the time taken by the optimized ILP, and averaging these ratios. In the "prep" and "vars" column, a dash is written only if no instance was solved with safety information. In the remaining columns a dash means not applicable.

In Table 1 we show the experimental results of the safety-optimized ILPs for MinPathError.(As explained before, for this problem it is correct to select $C = A$.) When the width of the graph is small, the speed-ups are also small. However, such graphs are fast to solve without safety in the first place. The challenging cases are the graphs of larger widths and more arcs (indeed, the solver with no safety information performs poorly and it becomes slower as the width increases). The speed-up of the solver significantly increases as the width of the graph increases, becoming more than 200$\times$ in some datasets. Moreover, these speed-up are obtained almost for free, in the sense that the time needed to compute safe sequences is negligible for all datasets, i.e. below 0.1 seconds, and often even much faster. We highlight the notable difference between the number of instances solved with no safety and with safety. In Table 2 for LeastSquares we observe the same phenomena, obtaining in some datasets speed-ups of 300$\times$ on graphs of larger widths, and solving always more instances with safety than without safety. In the extended version results for $C \subsetneq A$ in the MinPathError and for $C = A$ in the LeastSquares can be found.

References

1 Isoseqsim. URL: `https://github.com/yunhaowang/IsoSeqSim` [cited April 23, 2025].

2 Bashar Ahmed, Siddharth Singh Rana, Shahbaz Khan, et al. Evaluating Optimal Safe Flows Decomposition for RNA Assembly. *arXiv preprint arXiv:2409.13279*, 2024.

3 Alfred V Aho and Jeffrey D Ullman. *The theory of parsing, translation, and compiling*, volume 1. Prentice-Hall Englewood Cliffs, NJ, 1973.

4 Ravindra K Ahuja, Thomas L Magnanti, James B Orlin, et al. *Network flows: theory, algorithms, and applications*, volume 1. Prentice hall Englewood Cliffs, NJ, 1993.

5 Frances E Allen. Control flow analysis. *ACM Sigplan Notices*, 5(7):1–19, 1970. `doi:10.1145/800028.808479`.

6 Stephen Alstrup, Dov Harel, Peter W Lauridsen, and Mikkel Thorup. Dominators in linear time. *SIAM Journal on Computing*, 28(6):2117–2132, 1999. `doi:10.1137/S0097539797317263`.

7 Adam L Buchsbaum, Loukas Georgiadis, Haim Kaplan, Anne Rogers, Robert E Tarjan, and Jeffery R Westbrook. Linear-time algorithms for dominators and other path-evaluation problems. *SIAM Journal on Computing*, 38(4):1533–1573, 2008. `doi:10.1137/070693217`.

Table 1 Experimental results of MinPathError. Safety is with respect to path covers of $C = A$.

	w	#g	avg m (max m)	prep (s)	vars (%)	#solved (avg time (s))		×
						no safety	safety	
Zebrafish	1-3	15405	14 (210)	0.003	87.5	15405 (0.011)	15405 (0.011)	1.2
	4-6	239	35 (156)	0.007	29.5	239 (1.927)	239 (0.095)	14.6
	7-9	6	70 (158)	0.015	15.3	4 (71.070)	6 (0.645)	244.3
	10+	1	81 (81)	0.016	13.1	0 (-)	1 (3.421)	88.1
Human	1-3	10729	15 (331)	0.003	78.9	10729 (0.020)	10729 (0.014)	1.4
	4-6	947	35 (163)	0.008	24.7	944 (4.031)	947 (0.147)	19.0
	7-9	92	54 (105)	0.012	14.1	63 (63.346)	92 (3.924)	116.6
	10+	12	80 (104)	0.019	11.7	2 (189.233)	9 (32.264)	35.0
Mouse	1-3	12280	14 (128)	0.003	84.4	12280 (0.014)	12280 (0.011)	1.3
	4-6	749	35 (126)	0.007	27.4	749 (2.295)	749 (0.121)	15.0
	7-9	60	58 (231)	0.015	17.5	43 (52.717)	60 (4.955)	73.3
	10+	9	79 (121)	0.014	15.2	1 (262.851)	7 (34.163)	69.2
SRR020730	1-3	35069	9 (186)	0.002	84.7	35069 (0.015)	35069 (0.010)	1.4
	4-6	4497	33 (171)	0.007	20.4	4496 (2.043)	4497 (0.124)	12.7
	7-9	1008	52 (131)	0.010	11.9	844 (42.962)	1007 (1.692)	117.1
	10+	296	75 (184)	0.015	7.9	80 (63.971)	289 (22.945)	142.5
Mouse ONT	1-3	18527	11 (129)	0.003	81.4	18527 (0.013)	18527 (0.011)	1.3
	4-6	3083	31 (387)	0.007	30.2	3083 (0.203)	3083 (0.041)	4.8
	7-9	762	48 (157)	0.010	18.1	755 (2.495)	762 (0.180)	38.7
	10+	442	87 (589)	0.021	11.1	372 (18.194)	437 (0.611)	266.0
Mouse ONT.tr	1-3	18029	12 (131)	0.003	81.1	18029 (0.011)	18029 (0.011)	1.1
	4-6	3028	32 (187)	0.007	30.6	3028 (0.187)	3028 (0.042)	3.8
	7-9	803	50 (419)	0.012	19.0	796 (2.344)	803 (0.182)	21.9
	10+	476	89 (549)	0.022	11.5	403 (17.204)	472 (0.982)	162.1
Mouse PacBio	1-3	14256	14 (382)	0.003	83.5	14256 (0.010)	14256 (0.011)	1.2
	4-6	1376	33 (187)	0.006	33.3	1376 (0.196)	1376 (0.040)	5.2
	7-9	182	49 (159)	0.009	20.5	181 (4.083)	182 (0.144)	48.1
	10+	63	109 (1178)	0.026	13.3	53 (22.188)	62 (2.557)	210.3

8 Benjamin Merlin Bumpus, Bart M. P. Jansen, and Jari J. H. de Kroon. Search-space reduction via essential vertices. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 30:1–30:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ESA.2022.30.

9 Massimo Cairo, Shahbaz Khan, Romeo Rizzi, Sebastian Schmidt, Alexandru I. Tomescu, and Elia C. Zirondelli. A simplified algorithm computing all s-t bridges and articulation points. *Discrete Applied Mathematics*, 305:103–108, 2021. doi:10.1016/j.dam.2021.08.026.

10 Massimo Cairo, Romeo Rizzi, Alexandru I. Tomescu, and Elia C. Zirondelli. Genome Assembly, from Practice to Theory: Safe, Complete and Linear-Time. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.43.

Table 2 Experimental results of LeastSquares. Safety is with respect to path covers of the subset C of arcs whose weight is at least as large as the 25-th percentile of the weights of all arcs.

	w	#g	avg m (max m)	prep (s)	vars (%)	#solved (avg time (s))		×
						no safety	safety	
Zebrafish	1-3	15405	14 (210)	0.003	83.1	15405 (0.031)	15405 (0.023)	1.5
	4-6	239	35 (156)	0.007	18.1	224 (13.295)	238 (3.061)	15.9
	7-9	6	70 (158)	0.011	8.4	0 (-)	2 (62.840)	4.8
	10+	1	81 (81)	-	-	0 (-)	0 (-)	-
Human	1-3	10729	15 (331)	0.003	74.6	10729 (0.046)	10729 (0.031)	1.8
	4-6	947	35 (163)	0.007	16.3	862 (17.153)	930 (6.334)	12.9
	7-9	92	54 (105)	0.011	9.6	4 (69.195)	53 (71.207)	29.1
	10+	12	80 (104)	-	-	0 (-)	0 (-)	-
Mouse	1-3	12280	14 (128)	0.003	80.0	12280 (0.035)	12280 (0.026)	1.6
	4-6	749	35 (126)	0.007	16.9	696 (14.851)	744 (5.917)	12.4
	7-9	60	58 (231)	0.015	9.9	0 (-)	16 (64.253)	121.9
	10+	9	79 (121)	-	-	0 (-)	0 (-)	-
SRR020730	1-3	35069	9 (186)	0.002	83.2	35069 (0.026)	35069 (0.017)	1.7
	4-6	4497	33 (171)	0.007	15.5	4428 (13.082)	4494 (1.275)	16.7
	7-9	1008	52 (131)	0.011	8.9	269 (127.595)	899 (25.515)	82.7
	10+	296	75 (184)	0.015	6.8	1 (7.363)	104 (61.880)	68.0
Mouse ONT	1-3	18527	11 (129)	0.003	75.9	18527 (0.020)	18527 (0.017)	1.6
	4-6	3083	31 (387)	0.007	22.5	3077 (1.448)	3082 (0.310)	7.6
	7-9	762	48 (157)	0.010	13.5	693 (15.761)	754 (1.602)	116.2
	10+	442	87 (589)	0.018	8.5	222 (37.135)	405 (11.143)	301.0
Mouse ONT.tr	1-3	18029	12 (131)	0.003	74.7	18029 (0.020)	18029 (0.017)	1.5
	4-6	3028	32 (187)	0.007	21.7	3012 (1.693)	3022 (0.426)	7.1
	7-9	803	50 (419)	0.011	13.5	727 (14.958)	794 (2.087)	76.2
	10+	476	89 (549)	0.021	8.8	251 (32.631)	423 (8.422)	183.2
Mouse PacBio	1-3	14256	14 (382)	0.003	74.4	14256 (0.034)	14256 (0.025)	1.7
	4-6	1376	33 (187)	0.007	22.1	1356 (2.622)	1372 (1.189)	12.3
	7-9	182	49 (159)	0.010	14.0	146 (37.428)	174 (6.627)	137.8
	10+	63	109 (1178)	0.015	10.3	15 (110.413)	46 (17.775)	370.6

11 Massimo Cairo, Romeo Rizzi, Alexandru I. Tomescu, and Elia C. Zirondelli. Genome assembly, from practice to theory: Safe, complete and *Linear-Time*. *ACM Trans. Algorithms*, 20(1):4:1–4:26, 2024. `doi:10.1145/3632176`.

12 Stefan Canzar, Sandro Andreotti, David Weese, Knut Reinert, and Gunnar W Klau. CIDANE: comprehensive isoform discovery and abundance estimation. *Genome Biology*, 17:1–18, 2016.

13 Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 2016.

14 Keith D Cooper, Timothy J Harvey, and Ken Kennedy. A simple, fast dominance algorithm. *Software Practice & Experience*, 4(1-10):1–8, 2001.

15 Fernando H C Dias, Manuel Cáceres, Lucia Williams, Brendan Mumey, and Alexandru I Tomescu. A safety framework for flow decomposition problems via integer linear programming. *Bioinformatics*, 39(11):btad640, October 2023. `doi:10.1093/bioinformatics/btad640`.

16 Fernando H. C. Dias and Alexandru I. Tomescu. Accurate flow decomposition via robust integer linear programming. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pages 1–14, 2024. `doi:10.1109/TCBB.2024.3433523`.

17 Fernando HC Dias, Lucia Williams, Brendan Mumey, and Alexandru I Tomescu. Fast, flexible, and exact minimum flow decompositions via ILP. In *RECOMB 2022 – International Conference on Research in Computational Molecular Biology*, pages 230–245. Springer, 2022.

18 Michael R. Fellows, Lars Jaffke, Aliz Izabella Király, Frances A. Rosamond, and Mathias Weller. What is known about vertex cover kernelization? *CoRR*, abs/1811.09429, 2018. `arXiv:1811.09429`.

19 Jianxing Feng, Wei Li, and Tao Jiang. Inference of isoforms from short sequence reads. *Journal of Computational Biology*, 18(3):305–321, 2011. `doi:10.1089/CMB.2010.0243`.

20 Wojciech Fraczak, Loukas Georgiadis, Andrew Miller, and Robert E Tarjan. Finding dominators via disjoint set union. *Journal of Discrete Algorithms*, 23:2–20, 2013. `doi:10.1016/J.JDA.2013.10.003`.

21 Thomas Gatter and Peter F Stadler. Ryūtō: network-flow based transcriptome reconstruction. *BMC bioinformatics*, 20:1–14, 2019. `doi:10.1186/S12859-019-2786-5`.

22 Loukas Georgiadis, Luigi Laura, Nikos Parotsidis, and Robert E Tarjan. Loop nesting forests, dominators, and applications. In *Experimental Algorithms: 13th International Symposium, SEA 2014, Copenhagen, Denmark, June 29–July 1, 2014. Proceedings 13*, pages 174–186. Springer, 2014.

23 Loukas Georgiadis and Nikos Parotsidis. Dominators in directed graphs: a survey of recent results, applications, and open problems. *Proc. of ISCIM*, 13:15–20, 2013.

24 Loukas Georgiadis, Robert Tarjan, and Renato Werneck. Finding dominators in practice. *Journal of Graph Algorithms and Applications*, 10(1):69–94, 2006. `doi:10.7155/JGAA.00119`.

25 Andreas Grigorjew, Fernando H. C. Dias, Andrea Cracco, Romeo Rizzi, and Alexandru I. Tomescu. Accelerating ILP solvers for minimum flow decompositions through search space and dimensionality reductions. In Leo Liberti, editor, *22nd International Symposium on Experimental Algorithms, SEA 2024, July 23-26, 2024, Vienna, Austria*, volume 301 of *LIPIcs*, pages 14:1–14:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. `doi:10.4230/LIPICS.SEA.2024.14`.

26 Jiong Guo and Rolf Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007. `doi:10.1145/1233481.1233493`.

27 Rajiv Gupta. Generalized dominators and post-dominators. In *Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 246–257, 1992. `doi:10.1145/143165.143216`.

28 Saber Hafezqorani, Chen Yang, Theodora Lo, Ka Ming Nip, René L Warren, and Inanc Birol. Trans-NanoSim characterizes and simulates nanopore RNA-sequencing data. *GigaScience*, 9(6):giaa061, June 2020. `doi:10.1093/gigascience/giaa061`.

29 Bart M. P. Jansen and Ruben F. A. Verhaegh. Search-space reduction via essential vertices revisited: Vertex multicut and cograph deletion. In Hans L. Bodlaender, editor, *19th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2024, June 12-14, 2024, Helsinki, Finland*, volume 294 of *LIPIcs*, pages 28:1–28:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. `doi:10.4230/LIPICS.SWAT.2024.28`.

30 Shahbaz Khan, Milla Kortelainen, Manuel Cáceres, Lucia Williams, and Alexandru I. Tomescu. Improving RNA assembly via safety and completeness in flow decompositions. *J. Comput. Biol.*, 29(12):1270–1287, 2022. `doi:10.1089/CMB.2022.0261`.

31 Shahbaz Khan and Alexandru I. Tomescu. Optimizing Safe Flow Decompositions in DAGs. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 72:1–72:17, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPICS.ESA.2022.72`.

32 Kyle Kloster, Philipp Kuinke, Michael P O'Brien, Felix Reidl, Fernando Sánchez Villaamil, Blair D Sullivan, and Andrew van der Poel. A practical FPT algorithm for flow decomposition and transcript assembly. In *ALENEX 2018 – Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments*, pages 75–86. SIAM, 2018. `doi:10.1137/1.9781611975055.7`.

33 Jingyi Jessica Li, Ci-Ren Jiang, James B Brown, Haiyan Huang, and Peter J Bickel. Sparse linear modeling of next-generation mRNA sequencing (RNA-Seq) data for isoform discovery and abundance estimation. *Proceedings of the National Academy of Sciences*, 108(50):19867–19872, 2011.

34 Wei Li, Jianxing Feng, and Tao Jiang. IsoLasso: a LASSO regression approach to RNA-Seq based transcriptome assembly. In *RECOMB 2011 – International Conference on Research in Computational Molecular Biology*, pages 168–188. Springer, 2011.

35 Yen-Yi Lin, Phuong Dao, Faraz Hach, Marzieh Bakhshi, Fan Mo, Anna Lapuk, Colin Collins, and S Cenk Sahinalp. CLIIQ: Accurate comparative detection and quantification of expressed isoforms in a population. In *WABI 2012 – 12th International Workshop on Algorithms in Bioinformatics*, pages 178–189. Springer, 2012.

36 Juntao Liu, Ting Yu, Zengchao Mu, and Guojun Li. TransLiG: a de novo transcriptome assembler that uses line graph iteration. *Genome Biology*, 20:1–9, 2019.

37 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Kernelization - preprocessing with a guarantee. In Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors, *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 129–161. Springer, 2012. doi:10.1007/978-3-642-30891-8_10.

38 Aziz M Mezlini, Eric JM Smith, Marc Fiume, Orion Buske, Gleb L Savich, Sohrab Shah, Sam Aparicio, Derek Y Chiang, Anna Goldenberg, and Michael Brudno. iReckon: simultaneous isoform discovery and abundance estimation from RNA-seq data. *Genome Research*, 23(3):519–529, 2013.

39 Ralf Möhring. Algorithmic Aspects of Comparability Graphs and Interval Graphs. In Ivan Rival, editor, *Graphs and Order: the role of graphs in the theory of ordered sets and its applications*. D. Reidel Publishing Company, 1984.

40 Nidia Obscura Acosta, Veli Mäkinen, and Alexandru I Tomescu. A safe and complete algorithm for metagenomic assembly. *Algorithms for Molecular Biology*, 13:1–12, 2018. doi:10.1186/S13015-018-0122-7.

41 Renata Ochranová. Finding dominators. In *Foundations of Computation Theory: Proceedings of the 1983 International FCT-Conference Borgholm, Sweden, August 21–27, 1983 4*, pages 328–334. Springer, 1983. doi:10.1007/3-540-12689-9_115.

42 Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, 14(4):417–419, 2017. doi:10.1038/nmeth.4197.

43 Andrey Prjibelski. IsoQuant graphs from PacBio and ONT Mouse sequencing data, October 2024. doi:10.5281/zenodo.13987688.

44 Andrey D. Prjibelski, Alla Mikheenko, Anoushka Joglekar, Alexander Smetanin, Julien Jarroux, Alla L. Lapidus, and Hagen U. Tilgner. Accurate isoform discovery with IsoQuant using long reads. *Nature Biotechnology*, 41(7):915–918, 2023. doi:10.1038/s41587-022-01565-y.

45 Amatur Rahman and Paul Medvedev. Assembler artifacts include misassembly because of unsafe unitigs and underassembly because of bidirected graphs. *Genome Research*, 32(9):1746–1753, 2022.

46 Zhaleh Safikhani, Mehdi Sadeghi, Hamid Pezeshk, and Changiz Eslahchi. SSP: An interval integer linear programming for de novo transcriptome assembly and isoform discovery of RNA-seq reads. *Genomics*, 102(5-6):507–514, 2013.

47 Palash Sashittal, Chuanyi Zhang, Jian Peng, and Mohammed El-Kebir. Jumper enables discontinuous transcript assembly in coronaviruses. *Nature Communications*, 12(1):6728, 2021.

48 Sebastian Schmidt, Santeri Toivonen, Paul Medvedev, and Alexandru I. Tomescu. Applying the Safe-And-Complete Framework to Practical Genome Assembly. In Solon P. Pissis and Wing-Kin Sung, editors, *24th International Workshop on Algorithms in Bioinformatics (WABI 2024)*, volume 312 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:16, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.WABI.2024.8.

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

50 Li Song, Sarven Sabunciyan, and Liliana Florea. CLASS2: accurate and efficient splice variant annotation from RNA-seq reads. *Nucleic Acids Research*, 44(10):e98–e98, 2016.

51 Alexandru I. Tomescu, Anna Kuosmanen, Romeo Rizzi, and Veli Mäkinen. A novel min-cost flow method for estimating transcript expression with RNA-Seq. *BMC Bioinformatics*, 14(5):S15, 2013. doi:10.1186/1471-2105-14-S5-S15.

52 Alexandru I. Tomescu and Paul Medvedev. Safe and complete contig assembly via omnitigs. In Mona Singh, editor, *Research in Computational Molecular Biology*, pages 152–163, Cham, 2016. Springer International Publishing. doi:10.1007/978-3-319-31957-5_11.

53 Benedicte Vatinlen, Fabrice Chauvet, Philippe Chrétienne, and Philippe Mahey. Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. *European Journal of Operational Research*, 185(3):1390–1401, 2008. doi:10.1016/J.EJOR.2006.05.043.

54 Lucia Williams, Alexandru I. Tomescu, and Brendan Mumey. Flow decomposition with subpath constraints. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 20(1):360–370, 2023. doi:10.1109/TCBB.2022.3147697.

55 Zheng Xia, Jianguo Wen, Chung-Che Chang, and Xiaobo Zhou. NSMAP: a method for spliced isoforms identification and quantification from RNA-Seq. *BMC Bioinformatics*, 12:1–13, 2011.

56 Jin Zhao, Haodi Feng, Daming Zhu, and Yu Lin. MultiTrans: an algorithm for path extraction through Mixed Integer Linear Programming for transcriptome assembly. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 19(1):48–56, 2021. doi:10.1109/TCBB.2021.3083277.

57 Hongyu Zheng, Cong Ma, and Carl Kingsford. Deriving ranges of optimal estimated transcript expression due to nonidentifiability. *Journal of Computational Biology*, 29(2):121–139, 2022. Proceedings paper of RECOMB 2021. doi:10.1089/CMB.2021.0444.