Cut Paths and Their Remainder Structure, with Applications

Massimo Cairo
Department of Computer Science, University of Helsinki, Finland

Shahbaz Khan[1]
Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India

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

Sebastian Schmidt
Department of Computer Science, University of Helsinki, Finland

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

Elia C. Zirondelli
Department of Mathematics, University of Trento, Italy

Abstract

In a strongly connected graph $G = (V, E)$, a *cut arc* (also called *strong bridge*) is an arc $e \in E$ whose removal makes the graph no longer strongly connected. Equivalently, there exist $u, v \in V$, such that all u-v walks contain e. Cut arcs are a fundamental graph-theoretic notion, with countless applications, especially in reachability problems.

In this paper we initiate the study of *cut paths*, as a generalisation of cut arcs, which we naturally define as those paths P for which there exist $u, v \in V$, such that all u-v walks contain P as subwalk. We first prove various properties of cut paths and define their remainder structures, which we use to present a simple $O(m)$-time verification algorithm for a cut path ($|V| = n$, $|E| = m$).

Secondly, we apply cut paths and their remainder structures to improve several reachability problems from bioinformatics, as follows. A walk is called *safe* if it is a subwalk of every node-covering closed walk of a strongly connected graph. *Multi-safety* is defined analogously, by considering node-covering *sets* of closed walks instead. We show that cut paths provide *simple* $O(m)$-time algorithms verifying if a walk is safe or multi-safe. For multi-safety, we present the first linear time algorithm, while for safety, we present a simple algorithm where the state-of-the-art employed complex data structures. Finally we show that the simultaneous computation of remainder structures of all subwalks of a cut path can be performed in linear time, since they are related in a structured way. These properties yield an $O(mn)$-time algorithm outputting all maximal multi-safe walks, improving over the state-of-the-art algorithm running in time $O(m^2 + n^3)$.

The results of this paper only scratch the surface in the study of cut paths, and we believe a rich structure of a graph can be revealed, considering the perspective of a path, instead of just an arc.

2012 ACM Subject Classification Applied computing → Computational biology; Mathematics of computing → Paths and connectivity problems; Theory of computation → Graph algorithms analysis

Keywords and phrases reachability, cut arc, strong bridge, covering walk, safety, persistence, essentiality, genome assembly

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.17

Related Version *Previous Version*: `https://arxiv.org/abs/2210.07530`

[1] Most of the work by the author was done while he was affiliated to University of Helsinki.

Funding *Alexandru I. Tomescu*: This work was partially funded by the Academy of Finland (grants No. 322595, 328877). *Shahbaz Khan, Sebastian Schmidt and Alexandru I. Tomescu*: This work was partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 851093, SAFEBIO).

1 Introduction

1.1 Motivation

Connectivity problems are a fundamental aspect of graph theory and graph algorithms. In directed graphs, *cut arcs* (also known as *strong bridges*) are a basic structure to characterise the reachability properties of the graph. They are defined as arcs whose removal makes the graph no longer strongly connected, or equivalently, as arcs e such that there exists a pair of nodes u, v such that each u-v walk contains e. Cut arcs and the related strongly connected components are nowadays part of any lecture about graph theory. Moreover, significant work has been done to investigate the properties of a graph in relation to its cut arcs, e.g. by finding all cut arcs in linear time [14], and, after linear-time preprocessing, answering connectivity queries in constant time under the removal of any single arc [11]. These gave rise to further theoretical advances, and are used in algorithms to compute e.g. 2-vertex connected components in directed graphs [10] or 2-edge connected components in directed graphs [9]. The results are also useful for more practical works, like in the analysis of non-equilibrium biochemical reaction networks [22] or when analysing real-world graphs such as social networks or the world wide web [15]. Naturally, cut arcs play a crucial role in various practical networks, as they represent critical links in e.g. communication networks, road networks or transportation networks.

A natural generalisation of a cut arc is a *cut path*[2], similarly defined as a walk (not a single arc) W such that there exists a pair of nodes u, v such that each u-v walk has W as subwalk. While there is (to the best of our knowledge) no research around cut paths as such, they seem to be equally fundamental as cut arcs. On the practical side, one can view cut paths as representing critical *routes* through social networks, the world wide web, communication networks, road networks or transportation networks. Additionally, in practical applications, when nodes represent street crossings or network routers, cut paths imply critical *links* within these objects. On the more theoretical side, in this paper we show that cut paths are a useful tool in some reachability problems theoretically modelling the genome assembly problem in bioinformatics. In addition to exhibiting interesting properties on their own, cut paths also allows us to improve several of these results, as we discuss in Section 1.3.

1.2 Overview of cut paths

To give an overview of cut paths, we start with some basic definitions. A *graph* $G = (V, E)$ with n nodes and m arcs is directed and may have self-loops. For an arc $e = (u, v)$, we call $u = \textsc{tail}(e)$ its *tail* and $v = \textsc{head}(e)$ its *head*. Sets of nodes can induce subgraphs in the standard manner. A graph is *strongly connected* if each pair of nodes is connected by a directed path in both directions. A *strongly connected component (SCC)* is a maximal induced subgraph that is strongly connected. A *cut arc* is an arc that upon removal increases the number of strongly connected components in G.

[2] Note that walks that contain a cycle cannot be cut paths, hence the name *cut path* and not *cut walk*.

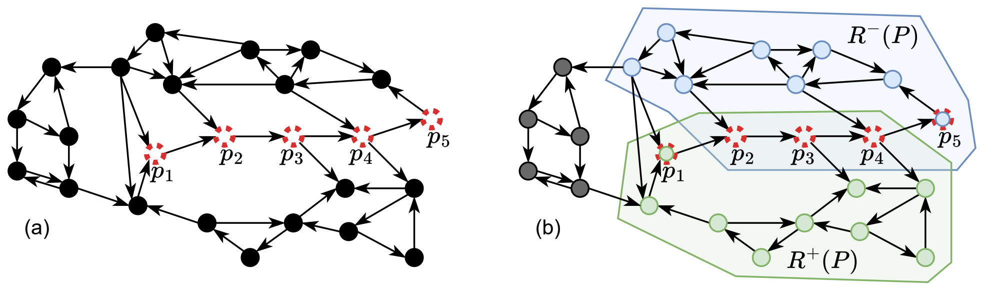

Figure 1 (a) A cut path $P = (p_1, p_2, p_3, p_4, p_5)$, highlighted by red dashed nodes. Each walk from p_1 to p_5 has P as a subwalk. (b) The remainder structure of P. The set $R^+(P)$ is enclosed by the green area, and the set $R^-(P)$ is enclosed by the blue area. Its source component $S^-(P)$ is highlighted by blue nodes, its sink component $S^+(P)$ is highlighted by green nodes and the inner component $\bar{S}(P)$ is highlighted by grey nodes. The path component $\bar{P}(P)$ is the intersection of $R^+(P)$ and $R^-(P)$. Note that, to get from $S^+(P)$ to $S^-(P)$, one needs to traverse P completely.

An equivalent definition of a cut arc in strongly connected graphs is an arc that is part of all walks from some node to some other node. Generalising, a *cut path* is a walk that is a subwalk of all walks from some node to some other node. We assume a graph to be strongly connected from here on.

When removing a cut arc (u, v) from a graph, the graph is not strongly connected anymore, but instead contains multiple SCCs. There is exactly one source SCC (containing v), which is an SCC without any incoming arcs from other SCCs, and exactly one sink SCC (containing u), which is an SCC without any outgoing arcs to other SCCs. The source is connected to the sink via direct arcs or via other SCCs.

A similar structure exists for cut paths, which we call their *remainder structure*. This structure is helpful both to efficiently check whether a walk is a cut path, and for applying cut paths to other problems. We define the remainder structure of a cut path $P = (p_1, \dots, p_\ell)$ (where p_i are nodes), which we denote $R(P) = (S^-(P), \bar{S}(P), S^+(P), \bar{P}(P))$, as follows. Let $R^+(P)$ be the set of nodes reachable by walks starting from the first node of P without using $(p_{\ell-1}, p_\ell)$. Symmetrically, let $R^-(P)$ be the set of nodes reaching the last node of P without using (p_1, p_2). With this definition, $R^+(P)$ and $R^-(P)$ may intersect, so we define the *source component* $S^-(P) := R^-(P) \setminus R^+(P)$, the *sink component* $S^+(P) := R^+(P) \setminus R^-(P)$, the *inner component* $\bar{S}(P) := V \setminus (S^- \cup S^+)$, and the *path component* $\bar{P}(P) := R^-(P) \cap R^+(P)$. In the remainder structure $R(P)$, for each node u in the subgraph induced by $S^-(P)$ and each node v in the subgraph induced by $S^+(P)$, the walk P is on every u-v walk in G. Computing the remainder structure is trivial given its definition, and it additionally allows for a very simple way to efficiently check if a walk is a cut path.

▶ **Theorem 1** (Efficient verification of cut paths). *Let $P = (p_1, \dots, p_\ell)$ be a walk of length $\ell - 1 \geq 1$. P is a cut path if and only if $\bar{P}(P) = \{p_2, \dots, p_{\ell-1}\}$ (specifically, for $\ell = 2$, $\bar{P}(P)$ is empty). This property can be verified in $O(m)$ time.*

1.3 Applications of cut paths

Background. We apply cut paths to improve several reachability-based results that have been given over the last years for "safe walks", motivated by the genome assembly problem in bioinformatics [20, 5, 6, 1, 17, 16]. We give here a minimal self-contained description, and refer the reader to these papers for motivation and applications. One can formulate the genome assembly problem as finding one closed node-covering walk (i.e., passing through every node at least once)[3] in a given strongly connected graph built from the input sequencing data. Since such graphs may admit multiple such walks, one can define a *safe walk* as one appearing in any closed node-covering walk of a strongly connected graph. Formally:

▶ **Definition 2** (Safe walk [20]). *Given a graph G, a walk is* safe *if it is a subwalk of each possible closed node-covering walk of G.*

We are interested in enumerating all *maximal* safe walks, namely all those that are not a proper subwalk of another safe walk. These can be thought as representing the maximal correct (partial) answers to the genome assembly problem. In [20] it is argued that some specific types of walks used by genome assembly programs are safe walks, and thus finding *all* maximal safe walks can be relevant in practice, since it can lead to longer parts of the genome being reconstructed. Similar problems have been previously studied without the covering constraint but instead by considering subwalks of all possible walks from a given node s to a given node t [4], or with the covering constraint set to *cover exactly once* (i.e., closed Eulerian walks) [16, 2].

Safe walks can be characterised as follows. Let $(w_1, \ldots, w_\ell)$ be a walk. A path from w_i to w_j, with $1 < i \leq j < \ell$, with first arc different from (w_j, w_{j+1}), and last arc different from (w_{i-1}, w_i), is called a *forbidden path*. Tomescu and Medvedev [20] proved that a walk is safe if and only if it has no forbidden path and all its arcs are cut arcs. For a walk made up only of cut arcs, such a forbidden path can be seen as a NO-certificate, since it testifies that the walk is not safe. Even though NO-certificates are usually harder to check, Cairo et al. [6] showed that the absence of a forbidden path can be checked in $O(m)$ time, but using complex data structures. Moreover, all maximal walks without forbidden paths can still be enumerated in $O(mn)$ time [5]. By appropriately splitting such walks at non-cut arcs, and removing duplicates, also maximal safe walks can be enumerated in $O(mn)$ time.[4]

One can also consider another variant of the problem, where one needs to assemble an unknown number of genomes from a graph, or a genome with an unknown number of chromosomes [1]. For this, one can formulate the genome assembly problem as finding a node-covering *set* of closed walks (i.e., such that every node appears in at least one walk in the set). In this setting, the notion of safety is adapted as follows:

▶ **Definition 3** (Multi-safe walk [20, 1]). *Given a graph G, a walk is* multi-safe *if it is a subwalk of some walk in each possible node-covering set of proper closed walks of G.*

The theory around multi-safe walks is less developed, the only algorithmic result being by Obscura Acosta et al. [1], who showed that all maximal multi-safe walks can be enumerated in $O(m^2 + n^3)$ time. This algorithm is also based on forbidden paths, with some additional conditions. One reason behind this lack of overall progress around multi-safe walks can be due to the lack of a YES-certificate, which requires building new machinery from scratch.

Safety-related previous works. The idea of partial solutions common to all solutions to a problem is very natural and has appeared in several other contexts. For example, Costa [7] studied *persistent edges* belonging to all maximum matching of a bipartite graph, and Hammer

[3] To be precise, [20, 5, 6, 1] focus mostly on the arc-covering case, where the closed walks have to pass through all *arcs* at least once. In this paper we focus on the node-covering case, for two reasons: first, it has a more direct relation to cut paths, and second, the arc-covering case can be reduced to it in linear time by subdividing every arc (i.e., introducing a node in the middle of every arc).

[4] This fact was not observed previously in the literature (recall that in this paper we are defining safety in terms of node-covering walks), but follows by standard techniques of removing duplicates using a suffix tree. For completeness, we explain this in Section 4 and Appendix A.

et al. [13] studied *persistent nodes* belonging to all maximum stable sets. Recently, Bumpus et al. [3] studied *c-essential vertices*, defined as those contained in all c-approximate solutions to e.g. Odd Cycle Transversal and Directed Feedback Vertex Set problems. See also Table 1 in [3] for algorithms detecting *some* c-essential vertices for several other NP-hard problems. As opposed to the latter problems, in this paper we tackle polynomially solvable problems (computing closed node-covering walks is trivial), and thus their safe partial solutions admit rich structures which can be exploited in getting efficient algorithms enumerating *all* of them.

Our results. We show that cut paths and their remainder structure provide a *flexible* technique to study both safe and multi-safe walks. For example, they can be used to derive natural YES-certificates for both types of walks.

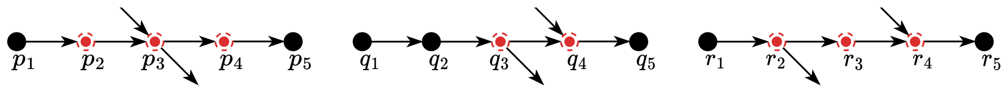

Figure 2 Walks P, Q, R and their cores highlighted by red dashed nodes. Nodes p_3, q_3 and r_2 are splits, and nodes p_3, q_4 and r_4 are joins. The walks P and Q are interleaved, and the walk R is non-interleaved. Note that $(q_1, \ldots, q_5)$ is defined to be interleaved, since even though it has separate split-free and join-free parts, they are not trivially safe since (q_3, q_4) is only safe if it is a cut arc.

To describe our results, we need additional definitions for walks. Examples for these definitions are given in Figure 2. A *split* is a node with at least two outgoing arcs and a *join* is a node with at least two incoming arcs. Let $W = (w_1, \ldots, w_\ell)$ be a walk with $\ell \geq 2$. The *inner* nodes of W are $w_2, \ldots, w_{\ell-1}$. Let w_i be its first inner join, or w_ℓ if W has no inner join. Let w_j be its last inner split, or w_1 if W has no inner split. Then W is an *interleaved walk* if $i \leq j+1$ and a *non-interleaved walk* otherwise. The *core* of an interleaved walk is its subwalk from w_{i-1} to w_{j+1}. The *core* of a non-interleaved walk is its subwalk from w_j to w_i.

A first consequence is that verifying whether a walk is safe can now be done by a simple check whether the *core* of a walk is a cut path (after excluding trivial cases). Since this can be computed in linear-time using simple graphs traversals (Theorem 1), we obtain a verification algorithm much simpler than the one in [6] (which uses complex data structures from [11], which in turn uses dominator trees [8] and loop-nesting forests [19]).

▶ **Theorem 4** (Safety characterisation and verification)**.** *Let W be a walk, and let $C(W)$ be its core. W is safe if and only if it is a non-interleaved walk or $C(W)$ is a cut path. This property can be verified in $O(m)$ time.*

In our proof, we use the properties of the remainder structure to show that if the core of a walk W is not a cut path, then it can be replaced by a walk avoiding the core in any closed node-covering walk. On the other hand, if the core is a cut path, then there is a pair of nodes that can only be connected via the core. Since a closed node-covering walk contains a subwalk between each pair of nodes, that makes W safe.

For the multi-safe case we obtain a YES-certificate based on checking a property of the remainder structure. This leads to the first linear-time algorithm verifying whether a walk is multi-safe. In contrast to safe walks, the characterisation depends on the existence of certain SCCs of size one. Intuitively, a multi-safe walk must be safe, since otherwise there would be a closed node-covering walk avoiding it, which then also disproves multi-safety. Moreover, if $R^+(C(W))$, $R^-(C(W))$ and $\bar{S}(C(W))$ of a core $C(W)$ contain only SCCs of size at least two, then they can all be covered by proper closed walks without leaving the respective component, and thus without using $C(W)$ as subwalk, disproving the multi-safety of W. If however one of them contains an SCC of size one, then to cover this SCC, the respective component needs to be left and reentered. This can only happen by using $C(W)$ as subwalk, hence W is multi-safe.

▶ **Theorem 5** (Multi-safety characterisation and verification). *Let W be a walk, and let $C(W)$ be its core. If W is non-interleaved, then it is multi-safe. Otherwise, it is multi-safe if and only if it is safe and any of $G[\bar{S}(C(W))], G[R^+(C(W))]$ or $G[R^-(C(W))]$ contains an SCC of size one. This property can be verified in $O(m)$ time.*

Lastly, we improve the existing $O(m^2 + n^3)$-time algorithm enumerating all maximal multi-safe walks. A naive application of Theorem 5 would lead to an $O(m^2 n)$-time algorithm for this enumeration problem, already improving the previous one for dense graphs. However, by proving several additional properties of the remainder structure, we can amortise the time to just $O(mn + o)$, where o is the size of the output. First, the remainder structure of all subwalks P' of a given cut path P can be precomputed in linear time. This works because when shifting either the start or end of the subwalk to the right (along the walk), then the set $R^+(P')$ monotonously grows and the set $R^-(P')$ monotonously shrinks (except for some subwalks that are trivial to handle without the remainder structure). Further, when growing $R^+(P')$, only complete SCCs get removed from the inner component, and when shrinking $R^-(P')$, the existing SCCs in the inner component are not altered.

▶ **Theorem 6** (Enumerating maximal multi-safe walks). *All maximal multi-safe walks can be identified in $O(mn)$ time and enumerated in $O(mn + o)$ time, where o is the total length of the output and it holds that $o \in O(n^3)$.*

In a nutshell, using cut paths and the remainder structure, we gain a deeper understanding of critical structures for the connectivity of directed graphs. In bioinformatics applications, this allows us to get better characterisations for two problems that for the first time admit a simple to compute and verify YES-certificate. Moreover, meticulously investigating the properties of the remainder structure, we improve over the complexity of the best known enumeration algorithm for maximal multi-safe walks.

Notation. A *split* is a node with at least two outgoing arcs, and a *join* is a node with at least two incoming arcs. Let $G = (V, E)$ be a strongly connected graph, with $|V| = n$ and $|E| = m \geq n$. We denote the removal of an arc $e \in E$ by $G - e$. Given a subset of nodes $V' \subseteq V$, the subgraph induced by V' subgraph is defined as $G[V'] = (V', \{(u, v) \in E \mid u, v \in V'\})$.

For two nodes $u, v \in V$, a *u-v walk* of *length* $\ell - 1$ is a sequence of nodes $W = (w_1, \ldots, w_\ell)$ with $w_1 = u$ and $w_\ell = v$ and such that for each $i \in \{1, \ldots, \ell - 1\}$ it holds that $(w_i, w_{i+1}) \in E$. The *tail* $\textsc{tail}(W)$ of W is w_1, and the *head* $\textsc{head}(W)$ of a W is w_ℓ. W is *closed* if $u = v$ and *open* otherwise. W is a *path* if all nodes are unique except that $w_1 = w_\ell$ is allowed. The *inner nodes* of a W are the nodes $w_2, \ldots, w_{\ell-1}$, where a walk of length $\ell - 1 \leq 1$ has no inner nodes. The notation WW' denotes the *concatenation* of walks W and W' if $\textsc{head}(W) = \textsc{tail}(W')$. *Subwalks* of walks are defined in the standard manner, where subwalks of closed walks may run over the end (e.g. (c, a, b) is a subwalk of (a, b, c, a)). A *proper subwalk* of W is a subwalk that is shorter than W.

2 Cut paths and their remainder structure

In this section, we formally define cut paths and the remainder structure and prove their main properties. See Appendix B for all formal proofs that we omitted here.

▶ **Definition 7.** *A walk W in a strongly connected graph $G = (V, E)$ is a* cut path *if there is a pair of nodes $u, v \in V$ such that all u-v walks in G have W as subwalk.*

One intuitive property of a cut path is that it is an open path.

▶ **Lemma 8** (Open path property). *A cut path is an open path.*

Further, for any cut path P, the pair of nodes $\textsc{tail}(P), \textsc{head}(P)$ is a *witness* for P being a cut path, i.e. all $\textsc{tail}(P)$-$\textsc{head}(P)$ walks contain P as a subwalk.

▶ **Lemma 9** (Witness property). *A walk W is a cut path if and only if it is subwalk of all $\textsc{tail}(W)$-$\textsc{head}(W)$ walks.*

2.1 Restricted reachabilities

The remainder structure is based on the restricted reachabilities of a walk. Even though only open paths can be cut paths, we define the remainder structure here on arbitrary walks. As we see below, the remainder structure makes it easy to check if a walk is a cut path.

▶ **Definition 10.** *The* restricted forward and backward reachability *of a walk $W = (w_1, \ldots, w_\ell)$ in a strongly connected graph $G = (V, E)$ are*

$$R^+(W) := \{v \in V \mid \exists\, \textsc{tail}(W)\text{-}v \text{ walk in } G - (w_{\ell-1}, w_\ell)\},$$
$$R^-(W) := \{v \in V \mid \exists\, v\text{-}\textsc{head}(W) \text{ walk in } G - (w_1, w_2)\}.$$

The restricted reachabilities exhibit the following property, which makes them simple to work with.

▶ **Lemma 11** (Bottleneck property). *For a walk $W = (w_1, \ldots, w_\ell)$, the only arc leaving $R^+(W)$ is $(w_{\ell-1}, w_\ell)$ and the only arc entering $R^-(W)$ is (w_1, w_2).*

Proof. Assume for a contradiction that an arc (u, v) different from $(w_{\ell-1}, w_\ell)$ would leave $R^+(W)$. Since $u \in R^+(W)$, there is a $\textsc{tail}(W)$-u walk without $(w_{\ell-1}, w_\ell)$. Appending (u, v) to such a walk cannot introduce $(w_{\ell-1}, w_\ell)$ as subwalk, because (u, v) is not $(w_{\ell-1}, w_\ell)$. Therefore, $v \in R^+(W)$, contradicting (u, v) leaving $R^+(W)$.

By symmetry, the only arc entering $R^-(W)$ is (w_1, w_2). ◀

Note that by leaving $R^+(W)$, one always ends up in $R^-(W)$ (or one was in $R^-(W)$ already, if one leaves $R^+(W)$ from a node in $R^+(W) \cap R^-(W)$), and by entering $R^-(W)$, one always comes from $R^+(W)$ (and possibly ends up in $R^+(W)$ again, if one enters $R^-(W)$ at a node in $R^+(W) \cap R^-(W)$). Further, the restricted reachabilities are strongly connected in certain cases.

▶ **Lemma 12** (Strong connectivity property). *Let $P = (p_1, \ldots, p_\ell)$ be a cut path. If the last inner node of P is a split, then $G[R^+(P)]$ is strongly connected. If the first inner node of P is a join, then $G[R^-(P)]$ is strongly connected.*

Proof. Let the last inner node of P be a split. By definition, p_1 reaches all nodes in $R^+(P)$ via walks not leaving $R^+(P)$. Assume for a contradiction that there is a node $v \in R^+(P)$ that cannot reach p_1 without leaving $R^+(P)$. Then by Lemma 11, each v-p_1 walk contains $(p_{\ell-1}, p_\ell)$, so each $p_{\ell-1}$-p_1 walk contains $(p_{\ell-1}, p_\ell)$. Let $v' \neq v_\ell$ be a node with $(p_{\ell-1}, v') \in E$. Then since v' is reachable from $p_{\ell-1}$, each v'-p_1 walk contains $(p_{\ell-1}, p_\ell)$. So there is a p_1-p_ℓ walk W via v' that does not have p_1 or p_ℓ as inner nodes, so it does not have P as subwalk. By Lemma 9, this contradicts P being a cut path. ◀

Note that, whenever a restricted reachability is not strongly connected, then it consists of a strongly connected component, plus nodes from P that form SCCs of size one.

2.2 The remainder structure

▶ **Definition 13.** *The* remainder structure $R(W) = (S^-(W), \bar{S}(W), S^+(W), \bar{P}(W))$ *of a walk W in a strongly connected graph $G = (V, E)$ is defined as*

$$\begin{aligned} S^-(W) &:= R^-(W) \setminus R^+(W) \text{ (the source component)},\\ \bar{S}(W) &:= V \setminus (R^+(W) \cup R^-(W)) \text{ (the inner component)},\\ S^+(W) &:= R^+(W) \setminus R^-(W) \text{ (the sink component)},\\ \bar{P}(W) &:= R^+(W) \cap R^-(W) \text{ (the path component)}. \end{aligned}$$

Note that the remainder structure is a decomposition of the nodes of G. For checking if a walk is a cut path, we can use the *inner path property* of the remainder structure. The inner path property can be checked in linear time with trivial algorithms that directly follow from the definition of the remainder structure and the property.

▶ **Theorem 1** (Efficient verification of cut paths). *Let $P = (p_1, \ldots, p_\ell)$ be a walk of length $\ell - 1 \geq 1$. P is a cut path if and only if $\bar{P}(P) = \{p_2, \ldots, p_{\ell-1}\}$ (specifically, for $\ell = 2$, $\bar{P}(P)$ is empty). This property can be verified in $O(m)$ time.*

Proof. By definition, it holds that $\{p_2, \ldots, p_{\ell-1}\} \subseteq \bar{P}(P)$. Assume for a contradiction that there was a node $v \in \bar{P}(P) \setminus \{p_2, \ldots, p_{\ell-1}\}$. If $v = p_\ell$, then $p_\ell \in R^+(P)$, so there is a $\text{TAIL}(P)$-$\text{HEAD}(P)$ walk that does contain $(p_{\ell-1}, p_\ell)$, which by Lemma 9 contradicts P being a cut path. In the same way, if $v = p_1$, then $p_1 \in R^-(P)$, which again contradicts P being a cut path. Therefore, $v \notin \{p_1, \ldots, p_\ell\}$.

Since $v \in R^+(P)$, there is a $\text{TAIL}(P)$-v walk W_1 in G that does not contain $(p_{\ell-1}, p_\ell)$. Further, since $v \in R^-(P)$, there is a v-$\text{HEAD}(P)$ walk W_2 in G that does not contain (p_1, p_2). Then, $W = W_1 W_2$ is a $\text{TAIL}(P)$-$\text{HEAD}(P)$ walk. Since $v \notin \{p_1, \ldots, p_\ell\}$, it holds that concatenating $W_1 W_2$ does not introduce P as subwalk. So by Lemma 9 it holds that P is not a cut path, which completes the contradiction.

If $\ell > m$, then W contains a cycle, so by Lemma 8 it is not a cut path. The sets $R^+(W)$ and $R^-(W)$ can be computed in linear time and hence $R^+(W) \cap R^-(W)$ can be computed in linear time. Resulting, P being a cut path can be verified in $O(m)$ time. ◀

The remainder structure exhibits two more properties useful for other problems.

▶ **Lemma 14** (Extended witness property). *Let $P = (p_1, \ldots, p_\ell)$ a cut path of length $\ell - 1 \geq 1$. Let $u \in S^+(P)$ and $v \in S^-(P)$ be nodes. It holds that all u-v walks contain P as subwalk.*

▶ **Lemma 15** (Nonemptiness property). *For a cut path $P = (p_1, \ldots, p_\ell)$ of length $\ell - 1 \geq 1$, it holds that $p_1 \in S^+(P)$ and $p_\ell \in S^-(P)$.*

3 Linear-time verifiable characterisations of (multi-)safe walks

We apply the remainder structure of a cut path to give easily and efficiently verifiable characterisations of safe and multi-safe walks. From here on we assume that our strongly connected graph $G = (V, E)$ is not a cycle. All missing formal proofs are in Appendix B.

First note that the univocal extension of a walk always needs to be traversed when traversing the walk itself with a closed walk. The *univocal extension* $U(W) = (l_1, \ldots, l_a, w_1, \ldots, w_\ell, r_1, \ldots, r_b)$ of W is a maximal walk where $l_2, \ldots, l_a, w_1$ are not joins and $w_\ell, r_1, \ldots, r_{b-1}$ are not splits.

▶ **Lemma 16** (Safety of univocal extensions). *The univocal extension $U(W)$ of a walk W is safe if and only if W is safe. It is multi-safe if and only if W is multi-safe.*

Since walks are (not necessarily maximal) univocal extensions of their cores, we get the following property useful for characterising safe and multi-safe walks.

▶ **Lemma 17** (Safety of cores). *The core $C(W)$ of a walk W is safe if and only if W is safe. It is multi-safe if and only if W is multi-safe.*

The difficulty in characterising safe and multi-safe walks lies in characterising interleaved walks. Non-interleaved walks are very simple to handle.

▶ **Lemma 18** (Non-interleaved walks). *A non-interleaved walk $W = (w_1, \dots, w_\ell)$ with core $C(W) = (w_j, \dots, w_i)$, $j + 2 \leq i$, is both safe and multi-safe. This property can be verified in $O(m)$ time.*

Safe walks can be characterised as follows.

▶ **Theorem 4** (Safety characterisation and verification). *Let W be a walk, and let $C(W)$ be its core. W is safe if and only if it is a non-interleaved walk or $C(W)$ is a cut path. This property can be verified in $O(m)$ time.*

Proof. If W is non-interleaved, the statement follows by Lemma 18.

If the core $C(W)$ of W is not a cut path, then by Lemma 9, there is a $\textsc{tail}(C(W))$-$\textsc{head}(C(W))$ walk that does not have $C(W)$ as subwalk. This can be used to replace all occurrences of $C(W)$ in a node-covering closed walk. To ensure that the resulting walk C covers all nodes, we insert two closed walks C_1 and C_2 into it, constructed as follows. Let v be the last inner split in $C(W)$ and v' one of its successors outside of $C(W)$. C_1 starts in $\textsc{tail}(C(W))$ and walks $C(W)$ until v and then v'. From v' it walks back to $\textsc{tail}(C(W))$, which is possible because the graph is strongly connected, and also possible without $C(W)$ as subwalk since it ends in $\textsc{tail}(C(W))$. C_2 is constructed symmetrically through $\textsc{head}(C(W))$. Since W is interleaved, it holds that $C(W)$ is interleaved, so C_1 and C_2 together cover $C(W)$. Further, because C contains $\textsc{tail}(C(W))$ and $\textsc{head}(C(W))$, we can insert C_1 and C_2 into it. Since the insertions happen at the first/last node of $C(W)$ they do not introduce it as subwalk. Thus, W is not safe.

If the core $C(W)$ of W is a cut path, then there is a pair of nodes $u, v \in V$ such that all u-v walks have $C(W)$ as a subwalk. A closed node-covering walk contains a subwalk between each pair of nodes, so also between u and v. Therefore, each closed node-covering walk contains $C(W)$ as subwalk, so by Lemma 17, W is safe.

Finally, the core $C(W)$ can be computed in linear time, and by Theorem 1 it can be checked for being a cut path in linear time. Hence, verifying if W is safe takes $O(m)$ time. ◀

Multi-safe walks can be characterised as follows. See Figure 3 for an example.

▶ **Theorem 5** (Multi-safety characterisation and verification). *Let W be a walk, and let $C(W)$ be its core. If W is non-interleaved, then it is multi-safe. Otherwise, it is multi-safe if and only if it is safe and any of $G[\bar{S}(C(W))], G[R^+(C(W))]$ or $G[R^-(C(W))]$ contains an SCC of size one. This property can be verified in $O(m)$ time.*

Proof. If W is non-interleaved, the statement follows by Lemma 18.

If $C(W)$ is not a cut path, then by Theorem 4, W is not safe, so it is also not multi-safe.

If none of $G[\bar{S}(C(W))], G[R^+(C(W))]$ and $G[R^-(C(W))]$ contain an SCC of size one, then all nodes can be covered by proper closed walks that do not leave a single one of $G[\bar{S}(C(W))], G[R^+(C(W))]$ or $G[R^-(C(W))]$. By definition, $G[\bar{S}(C(W))]$ contains no

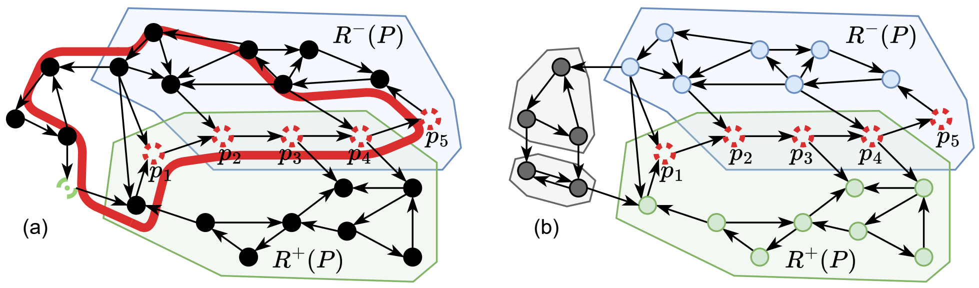

Figure 3 (a) A walk $(p_1, p_2, p_3, p_4, p_5)$ (as red dashed nodes) that is multi-safe, because the inner component of its remainder structure contains an SCC of size one (the green dashed node). The red walk is an example proper closed walk that covers the marked SCC. (b) Neither the restricted reachabilities nor the inner component contain an SCC of size one, so the two SCCs in the inner component and the restricted reachabilities can be covered by separate closed walks, without traversing the walk.

node of $C(W)$. Further, by Lemma 11, $G[R^+(C(W))]$ misses the last node of $C(W)$ and $G[R^-(C(W))]$ misses the first node of $C(W)$. Thus, none of the proper closed walks can have $C(W)$ as subwalk, so W is not multi-safe.

If $C(W)$ is a cut path and $G[\bar{S}(C(W))]$ contains an SCC of size one, then this SCC cannot be covered by a proper closed walk without leaving $\bar{S}(C(W))$. By Lemmas 11 and 15, when leaving $\bar{S}(C(W))$, a walk ends up in $S^+(C(W))$, and when entering $\bar{S}(C(W))$, a walk comes from $S^-(C(W))$. By Lemma 14, walking from $S^+(C(W))$ to $S^-(C(W))$ requires having $C(W)$ as a subwalk. Therefore, by Lemma 17, W is multi-safe.

If $C(W)$ is a cut path and $G[R^+(C(W))]$ contains an SCC of size one, then by Lemma 12, $R^+(C(W))$ contains exactly one node v. By definition, v is not contained in $G[\bar{S}(C(W))]$ and by Lemma 15, v is not contained in $G[R^-(C(W))]$. So to cover v, a proper closed walk X would leave $R^+(C(W))$. By Lemmas 11 and 15, this implies that X would have a subwalk from $\text{TAIL}(C(W))$ to $\text{HEAD}(C(W))$. By Lemma 9 this implies X has $C(W)$ as subwalk, so by Lemma 17, W is multi-safe. By symmetry, if $C(W)$ is a cut path and $G[R^-(C(W))]$ contains an SCC of size one, then W is multi-safe.

Finally, the core $C(W)$ can be computed in linear time, and by Theorem 1 it can be checked for being a cut path in linear time. Moreover, the strongly connected components can be computed in linear time by [18]. Hence, W being safe can be verified in $O(m)$ time. ◀

4 Amortised enumeration of all maximal multi-safe walks

In this section we give the algorithm supporting Theorem 6. Since every multi-safe walk is also safe, by definition, we start by enumerating all safe walks, and then finding their subwalks that are also multi-safe, using further properties of their remainder structure, including an additional monotonicity property of it.

Enumerating all maximal safe walks. Recall that all walks without forbidden paths can be enumerated in time $O(mn)$ with the algorithm from Cairo et al. [5]. From these, it is simple to get the safe walks using the following property from [20, Theorem 3]:

▶ **Lemma 19** (Safe walks [20]). *A walk is safe if and only if it has no forbidden paths and has no cut arc.*

Algorithm 1 MultiSafe.

```
Input: Strongly connected graph $G = (V, E)$, all maximal safe walks $\mathcal{W}$.
Output: All maximal multi-safe walks $\mathcal{W}'$.
1  $\mathcal{W}' \leftarrow ()$ // empty list
2  for $W \in \mathcal{W}$ do
3      if $W$ is a non-interleaved walk then
4          append $W$ to $\mathcal{W}'$, continue
5      $(w_1, \dots, w_\ell) \leftarrow C(W)$, $start \leftarrow 1$, $end \leftarrow 1$
6      while $end \leq \ell$ do
7          if $(w_{start}, \dots, w_{end})$ is multi-safe then
8              $end \leftarrow end + 1$
9          else
10             append $U((w_{start}, \dots, w_{end-1}))$ to $\mathcal{W}'$
11             $start \leftarrow start + 1$
12     while $(w_{start}, \dots, w_{end-1})$ is not multi-safe do
13         $start \leftarrow start + 1$
14     append $U((w_{start}, \dots, w_{end-1}))$ to $\mathcal{W}'$
15 Remove duplicates and subwalks from $\mathcal{W}'$ using e.g. a suffix tree
```

We compute the cut arcs in linear time [14] and then break all the walks without forbidden paths at arcs that are not cut arcs. Then we remove duplicates and proper subwalks from the result using standard methods and suffix trees. See Appendix A for details.

▶ **Lemma 20** (Enumeration of safe walks). *All maximal safe walks can be enumerated in $O(mn)$ time.*

Enumerating all maximal multi-safe walks. Using Theorem 5, we are able to derive an algorithm that enumerates maximal multi-safe walks. It works by iterating all safe walks and enumerating all their maximal multi-safe subwalks. We start with the subwalk of a safe walk consisting of its first core arc, and then extend it to the right whenever it is safe, while removing its first node (only from the subwalk, not the graph) whenever it is not safe. Afterwards, we deduplicate and remove proper subwalks again, as described in Appendix A. See Algorithm 1 for pseudocode. To analyse the runtime of this algorithm without amortisation, we use the following property.

▶ **Lemma 21** (Amount of interleaved safe walks). *A strongly connected graph contains at most $O(n)$ interleaved maximal safe walks.*

By Cairo et al. [5], the total length of the maximal walks without forbidden paths is $O(mn)$, and hence the total length of the maximal safe walks is $O(mn)$. Thus, if there are no interleaved walks, the algorithm runs in $O(mn)$ time. However, there may be up to $O(n)$ interleaved walks, and since their cores can not have cycles, for each interleaved walk, Algorithm 1 performs up to $O(n)$ multi-safety checks. Each such check takes $O(m)$ by Theorem 5. Further, in a strongly connected graph that is not a cycle, a univocal extension increases the length of a walk by at most $O(n)$. So the total length of the maximal multi-safe walks produced by a maximal safe walk is $O(n^2)$. Hence, including linear-time deduplication and removal of proper subwalks, the runtime of Algorithm 1 is $O(mn)$ for non-interleaved walks plus $O(mn + n^2)$ for each of the up to $O(n)$ interleaved walks. Summed up, that is $O(mn^2 + n^3) = O(mn^2)$. However, using amortisation, we get $O(mn)$, as shown below.

▶ **Theorem 22** (Amortised computation). *Let $P = (p_1, \ldots, p_\ell)$ be a cut path and let $P' = (p_i, \ldots, p_j)$ be a subwalk of P with $i < j$. If p_{j-1} and $p_{\ell-1}$ are splits, then $R^+(P') \subseteq R^+(P)$. If p_2 and p_{i+1} are joins, then $R^-(P') \subseteq R^-(P)$.*

Proof. Let p_{j-1} and $p_{\ell-1}$ be splits. Let $v \notin R^+(P)$. Then by definition each p_1-v walk contains $(p_{\ell-1}, p_\ell)$. Further, since $i \neq \ell$ and by Lemma 8 it holds that P is an open path, it holds that $p_i \in R^+(P)$. So each p_i-v walk contains $(p_{\ell-1}, p_\ell)$. Further, since P is a cut path, also its subwalk $P_C = (p_i, \ldots, p_{\ell-1})$ is a cut path. By Lemma 9, this implies that each p_i-$p_{\ell-1}$ walk has P_C as subwalk, which especially means that it contains (p_{j-1}, p_j). Therefore, each p_i-v walk contains (p_{j-1}, p_j), so $v \notin R^+(P')$. Resulting, $R^+(P') \subseteq R^+(P)$.

By symmetry, if p_2 and p_{i+1} are joins, then $R^-(P') \subseteq R^-(P)$. ◀

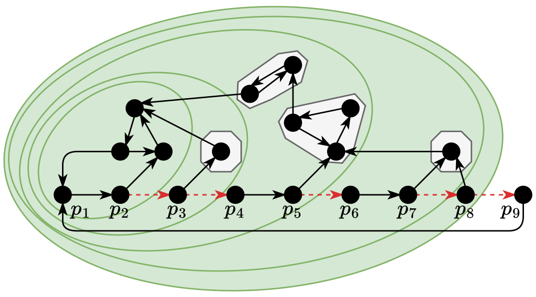

Figure 4 Amortised computation of $R^+(P')$ where P' is a prefix of $P = (p_1, \ldots, p_9)$. The red dashed arcs mark the ends of the prefixes P'. They are also the only arc leaving their respective $R^+(P')$. The SCCs of $G \setminus R^+(P')$ are enclosed in grey areas, and the different $R^+(P')$ are enclosed in green areas.

An example for Theorem 22 is given in Figure 4. Using Theorem 22 to implement Algorithm 1, we can answer all multi-safety queries for a single walk $W \in \mathcal{W}$ in $O(m)$ time. For this, we observe that the boundaries of the subwalk W' of $C(W)$ that is tested for safety only get shifted towards the end of $C(W)$. Further, whenever W' contains no split or no join as inner node, then it is either a non-interleaved walk or a univocal extension of a cut arc (by Lemma 19) and hence safe. So we only need to compute the remainder structure for subwalks that contain at least one split and one join. For such subwalks, by Theorem 22 it holds that $R^+(W')$ is monotonically increasing within each execution of the body of the loop in Algorithm 1, and $R^-(W')$ is monotonically decreasing. Therefore, all $R^+(W')$ and $R^-(W')$ can be precomputed by computing them for all prefixes and suffixes of W that contain splits or joins, respectively. The computation of the $R^+(W')$ is done in forward order, and for each node, the search is started from the node itself and nodes visited by earlier searches are pruned. With this strategy, the $R^+(W')$ of all prefixes W' of $C(W)$ can be computed in $O(m)$ time. By computing the $R^-(W')$ in reverse order, they can also be computed in $O(m)$ time.

To check the safety of all subwalks W' based on the precomputed remainder structure in linear time, note the following. When executing the body of the loop in Algorithm 1, $R^+(W')$ only increases and $R^-(W')$ only decreases for the relevant subwalks (those that are interleaved). Further, by Lemma 12, $G[R^+(W')]$ and $G[R^-(W')]$ are strongly connected if they have a split as last inner node or a join as first inner node, respectively. If they are not strongly connected then the inner nodes after the last inner split or the inner nodes before the first inner join, respectively, form SCCs of size 1. Therefore, the query if $G[R^+(W')]$ or $G[R^-(W')]$ contain an SCC of size 1 can be answered in constant time if the size of $R^+(W')$

and $R^-(W')$ as well as the joins and splits of W' are tracked within the body of the loop in Algorithm 1. Further, by definition of $R^+(W')$ and $R^-(W')$, when $R^+(W')$ grows, SCCs of the inner component enter it as a whole, so by growing $R^+(W')$, the remaining SCCs of the inner component remain unchanged. Symmetrically, when shrinking $R^-(W')$, the new nodes do not alter the existing SCCs in the inner component, so to check whether an SCC of size one was added, only the SCCs of the induced subgraph of the newly added nodes need to be computed. Since the SCCs of a graph can be computed in linear time [18], this means that tracking whether there are SCCs of size 1 in any of $G[\bar{S}(W')]$, $G[R^+(W')]$ or $G[R^-(W')]$ can be implemented in $O(m)$ time per execution of the body of the loop in Algorithm 1.

Hence, we get a runtime of $O(m)$ for the multi-safety checks of each of the $O(n)$ interleaved safe walks. However, each interleaved maximal safe walk may produce maximal multi-safe walks of total length $O(n^2)$. But, amortising over all interleaved maximal safe walk, we see that each core of a multi-safe walk is uniquely identified by its first and last node, since otherwise it would not be a cut path by Lemma 9. So, we can use a flag for each pair of nodes and thus avoid repetitions in multi-safe walks produced by interleaved maximal safe walks in constant time. In total, at most $O(n^2)$ such checks happen, so the interleaved walks take $O(mn + o)$ time, where o is the total length of the interleaved walks. This results in a total time of $O(mn+o)$ for Algorithm 1, and since there are at most n interleaved safe walks by Lemma 21, $o \in O(n^3)$. By reporting the maximal multi-safe walks as start and end index in their respective maximal safe walks, we get an output size of $O(n^2)$ interleaved maximal multi-safe walks, plus $O(mn)$ non-interleaved maximal multi-safe walks. So if we are only interested in identifying maximal multi-safe walks and not in an explicit enumeration, we have an algorithm that runs in $O(mn)$ time.

▶ **Theorem 6** (Enumerating maximal multi-safe walks). *All maximal multi-safe walks can be identified in $O(mn)$ time and enumerated in $O(mn + o)$ time, where o is the total length of the output and it holds that $o \in O(n^3)$.*

5 Conclusions and future work

We introduced cut paths as a generalisation of cut arcs, as well as the remainder structure of cut paths. Using properties of the remainder structure, we applied cut paths to some well-studied reachability problems from bioinformatics. In the same way as the remainder structure gave a simple YES-certificate for a path to be a cut path (Theorem 1), the remainder structure led to easily verifiable YES-certificates for walk safety (Theorem 4) and multi-safety (Theorem 5), which were open questions. By proving an additional monotonicity property (Theorem 22), we improved the state-of-the-art of enumeration of all maximal multi-safe walks (Theorem 6).

There are central structural questions about cut paths that remain open. It is known that there are at most $O(n)$ cut arcs which can be enumerated in $O(m)$ time [14]. But for cut paths, there is no known upper bound to their amount or total length, and it is open how they can be enumerated efficiently. Further, it is open how they can overlap and intersect.

For our applications, it is open if the total length of safe and multi-safe walks is really $O(mn)$, or if our enumeration algorithm for safe and multi-safe walks is not optimal. Further, it is open if there is a linear output-sensitive algorithm for safe or multi-safe walks, as the one for walks without forbidden paths from [6].

References

1 Nidia Obscura Acosta, Veli Mäkinen, and Alexandru I. Tomescu. A safe and complete algorithm for metagenomic assembly. *Algorithms for Molecular Biology*, 13(1):3:1–3:12, 2018. doi:10.1186/s13015-018-0122-7.

2 Nidia Obscura Acosta and Alexandru I. Tomescu. Simplicity in eulerian circuits: Uniqueness and safety. *arXiv*, abs/2208.08522, 2022. arXiv:2208.08522.

3 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)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:15, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2022.30.

4 Massimo Cairo, Shahbaz Khan, Romeo Rizzi, Sebastian Schmidt, and Alexandru I. Tomescu. Safety in s-t Paths, Trails and Walks. *Algorithmica*, 84:719–741, 2022. doi:10.1007/s00453-021-00877-w.

5 Massimo Cairo, Paul Medvedev, Nidia Obscura Acosta, Romeo Rizzi, and Alexandru I. Tomescu. An Optimal $O(nm)$ Algorithm for Enumerating All Walks Common to All Closed Edge-covering Walks of a Graph. *ACM Trans. Algorithms*, 15(4):48:1–48:17, 2019. doi:10.1145/3341731.

6 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, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 43:1–43:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

7 Marie Costa. Persistency in maximum cardinality bipartite matchings. *Oper. Res. Lett.*, 15(3):143–9, 1994. doi:10.1016/0167-6377(94)90049-3.

8 Wojciech Fraczak, Loukas Georgiadis, Andrew Miller, and Robert E Tarjan. Finding dominators via disjoint set union. *Journal of Discrete Algorithms*, 23:2–20, 2013.

9 Loukas Georgiadis, Giuseppe F Italiano, Luigi Laura, and Nikos Parotsidis. 2-edge connectivity in directed graphs. *ACM Transactions on Algorithms (TALG)*, 13(1):1–24, 2016.

10 Loukas Georgiadis, Giuseppe F Italiano, Luigi Laura, and Nikos Parotsidis. 2-vertex connectivity in directed graphs. *Information and Computation*, 261:248–264, 2018.

11 Loukas Georgiadis, Giuseppe F Italiano, and Nikos Parotsidis. Strong connectivity in directed graphs under failures, with applications. *SIAM Journal on Computing*, 49(5):865–926, 2020.

12 Dan Gusfield. Algorithms on stings, trees, and sequences: Computer science and computational biology. *Acm Sigact News*, 28(4):41–60, 1997.

13 P. L. Hammer, P. Hansen, and B. Simeone. Vertices belonging to all or to no maximum stable sets of a graph. *SIAM Journal on Algebraic Discrete Methods*, 3(4):511–522, 1982. doi:10.1137/0603052.

14 Giuseppe F Italiano, Luigi Laura, and Federico Santaroni. Finding strong bridges and strong articulation points in linear time. *Theoretical Computer Science*, 447:74–84, 2012.

15 Giuseppe F Italiano, Nikos Parotsidis, and Eugenia Perekhodko. What's inside a bow-tie: Analyzing the core of the web and of social networks. In *Proceedings of the 2017 International Conference on Information System and Data Mining*, pages 39–43, 2017.

16 Niranjan Nagarajan and Mihai Pop. Parametric complexity of sequence assembly: theory and applications to next generation sequencing. *Journal of computational biology*, 16(7):897–908, 2009.

17 Amatur Rahman and Paul Medvedev. Assembler artifacts include misassembly because of unsafe unitigs and underassembly because of bidirected graphs. *Genome Research*, 2022. doi:10.1101/gr.276601.122.

18 Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

19 Robert Endre Tarjan. Edge-disjoint spanning trees and depth-first search. *Acta Informatica*, 6(2):171–185, 1976.

20 Alexandru I. Tomescu and Paul Medvedev. Safe and complete contig assembly through omnitigs. *Journal of Computational Biology*, 24(6):590–602, 2017. Preliminary version appeared in RECOMB 2016.

21 Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

22 Pencho Yordanov and Jörg Stelling. Efficient manipulation and generation of kirchhoff polynomials for the analysis of non-equilibrium biochemical reaction networks. *Journal of the Royal Society Interface*, 17(165):20190828, 2020.

A Deduplication and removal of proper subwalks in linear time

Algorithm 2 RemoveDuplicatesAndProperSubwalks.

```
Input: List of walks $\mathcal{W} = (W_1, \dots, W_{|\mathcal{W}|})$.
Output: Set of walks $\mathcal{W}'$ containing one copy of each unique walk in $\mathcal{W}$ that is not
        a proper subwalk of another walk in $\mathcal{W}$.
1 Sort $\mathcal{W}$ by length descending
2 Build string $S = W_1\$W_2\$\dots\$W_{|\mathcal{W}|}\$$
3 Build suffix tree $T$ on $S$
4 $\mathcal{W} \leftarrow \emptyset$
5 for $W_i \in \mathcal{W}$ do
6     $(l, r) \leftarrow$ first occurrence of $W_i$ in $S$
7     if $(l, r) =$ coordinates of $W_i$ in $S$ then
8         $\mathcal{W}' \leftarrow \mathcal{W}' \cup \{W_i\}$
```

The removal of duplicates and subwalks is implemented in linear time using a suffix tree in Algorithm 2. It works by sorting the walks by length descending and only reporting a walk if it is no subwalk of a previous walk, meaning if it is no proper subwalk of a previous walk and it is the first occurrence of itself.

▶ **Lemma 23** (Deduplication). *Algorithm 2 is correct and works in time linear in the total length of $\mathcal{W}$.*

Proof. The algorithm sorts the walks by length descending and then reports walks only if their string of nodes does not occur any earlier than themselves in S. This reports only one copy of each walk since only the first occurrence of a walk in S is reported. Moreover, if a walk is a proper subwalk of another, then that subwalk will occur earlier in S, so the subwalk will never be reported. Therefore, Algorithm 2 is correct.

For the runtime, let $||\mathcal{W}||$ be the total length of $\mathcal{W}$. Sorting $\mathcal{W}$ by length descending can be done by bucket sort with $||\mathcal{W}||$ buckets containing each a dynamic array. Then it runs in time linear in $||\mathcal{W}||$. Building S and the suffix tree is linear in $||\mathcal{W}||$ [21]. The total cost of checking for the first occurrences of all W_i in S is linear in $||\mathcal{W}||$ [12]. Checking the coordinates of W_i can be done by storing the coordinates for each string while constructing S in a lookup table indexed by i. Then the branch runs in constant time. Therefore, Algorithm 2 runs in time linear in $||\mathcal{W}||$. ◀

B Omitted proofs

This section contains the proofs omitted from the main matter.

▶ **Lemma 8** (Open path property). *A cut path is an open path.*

Proof. Let W be a walk in a strongly connected graph $G = (V, E)$ that is not an open path, i.e. it repeats some node v. Then we can construct the walk $W' \neq W$ by removing all v-v subwalks from W (if W is a v-v walk, then W' is a single node). Assume for a contradiction that W was a cut path. Then there would be a pair of nodes $u, w \in V$ such that every u-w walk in G would have W as subwalk. But we could replace all occurrences of W by W' in any u-w walk, resulting in u-w walks that do not have W as subwalk. By contradiction, W is not a cut path. ◀

▶ **Lemma 9** (Witness property). *A walk W is a cut path if and only if it is subwalk of all $\textsc{tail}(W)$-$\textsc{head}(W)$ walks.*

Proof. Let $G = (V, E)$ be the strongly connected graph that contains W. If there is a $\textsc{tail}(W)$-$\textsc{head}(W)$ walk without W as subwalk, then for any pair of nodes $u, v \in V$, any u-v walk that contains W as subwalk can be transformed into a u-v walk that does not contain W as subwalk. Then W is not a cut path.

If all $\textsc{tail}(W)$-$\textsc{head}(W)$ walks have W as subwalk, then by definition W is a cut path. ◀

▶ **Lemma 14** (Extended witness property). *Let $P = (p_1, \ldots, p_\ell)$ a cut path of length $\ell - 1 \geq 1$. Let $u \in S^+(P)$ and $v \in S^-(P)$ be nodes. It holds that all u-v walks contain P as subwalk.*

Proof. Let $W_1W_2W_3$ be a $\textsc{tail}(P)$-$\textsc{head}(P)$ walk where W_1 is a $\textsc{tail}(P)$-u walk, W_2 is a u-v walk, and W_3 is a v-$\textsc{head}(P)$ walk. By Lemma 9, $W_1W_2W_3$ must have P as a subwalk since P is a cut path. By definition, $\textsc{tail}(P)$ reaches all nodes in $S^+(P) \subseteq R^+(P)$ without using $(p_{\ell-1}, p_\ell)$, so we can choose W_1 without using P as subwalk. Also, all nodes in $S^-(P) \subseteq R^-(P)$ reach $\textsc{head}(P)$ without using (p_1, p_2), so we can choose W_3 without using P as subwalk. Further, by definition, $u, v \notin \bar{P}(P)$, so by Theorem 1, neither u nor v are inner nodes of P. Therefore, concatenating $W_1W_2W_3$ cannot introduce P as a subwalk by crossing the boundary between either W_1 and W_2 or W_2 and W_3. Concluding, W_2 has P as subwalk. ◀

▶ **Lemma 15** (Nonemptiness property). *For a cut path $P = (p_1, \ldots, p_\ell)$ of length $\ell - 1 \geq 1$, it holds that $p_1 \in S^+(P)$ and $p_\ell \in S^-(P)$.*

Proof. By definition, $p_1 \in R^+(P)$ and $p_\ell \in R^-(P)$. It holds that $p_\ell \notin R^+(P)$ and $p_1 \notin R^-(P)$, since by Lemma 11 any of the two implies a $\textsc{tail}(P)$-$\textsc{head}(P)$ walk without P as subwalk, which by Lemma 9 contradicts P being a cut path. ◀

▶ **Lemma 16** (Safety of univocal extensions). *The univocal extension $U(W)$ of a walk W is safe if and only if W is safe. It is multi-safe if and only if W is multi-safe.*

Proof. Let $U(W) = (u_1, \ldots, u_\ell)$ and $W = (u_i, \ldots, u_j)$. Any closed walk having W as subwalk can only enter W via $(u_1, \ldots, u_i)$ since $(u_2, \ldots u_i)$ has no joins, and it can only leave W via $(u_j, \ldots, u_\ell)$ since $(u_j, \ldots, u_{\ell-1})$ has no splits. Hence, $U(W)$ is safe if and only if W is safe, and $U(W)$ is multi-safe if and only if W is multi-safe. ◀

Since walks are (not necessarily maximal) univocal extensions of their cores, we get the following property useful for characterising safe and multi-safe walks.

▶ **Lemma 17** (Safety of cores). *The core $C(W)$ of a walk W is safe if and only if W is safe. It is multi-safe if and only if W is multi-safe.*

Proof. Let $W = (w_1, \dots, w_\ell)$ and $C(W) = (w_i, \dots, w_j)$. By definition, independent of W being interleaved or non-interleaved, it holds that $(w_2, \dots, w_i)$ contains no joins and $(w_j, \dots, w_{\ell-1})$ contains no splits. Hence, W is subwalk of $U(C(W))$, so $U(W) = U(C(W))$. By Lemma 16, W is safe $\iff$ $U(W)$ is safe $\iff$ $U(C(W))$ is safe $\iff$ $C(W)$ is safe. The same equivalence holds for the multi-safe property. ◀

▶ **Lemma 18** (Non-interleaved walks). *A non-interleaved walk $W = (w_1, \dots, w_\ell)$ with core $C(W) = (w_j, \dots, w_i)$, $j + 2 \leq i$, is both safe and multi-safe. This property can be verified in $O(m)$ time.*

Proof. Since $j + 2 \leq i$, it holds that $C(W)$ has an inner node that can only be covered by a closed walk by using $C(W)$ as subwalk. Hence, by Lemma 17, it holds that W is both safe and multi-safe.

Finally, checking a walk for being interleaved can be done in $O(m)$ time. If $\ell > 3m$, then if the graph is a cycle, W is non-interleaved. If the graph is not a cycle, then assume for a contradiction that W is non-interleaved. Then W contains a join-free or split-free subwalk of length $m + 1$. Such a subwalk contains a cycle, because $m + 1 > m$. And such a cycle is then join-free or split-free. This contradicts the graph not being a cycle or the graph being strongly connected. ◀

▶ **Lemma 21** (Amount of interleaved safe walks). *A strongly connected graph contains at most $O(n)$ interleaved maximal safe walks.*

Proof. Note that each interleaved walk with a core of length 1 is safe only if its core is a cut arc. So there are at most $O(n)$ interleaved safe walks with a core of length 1. Further, by Cairo et al. [6], it holds that there are at most $O(n)$ walks without forbidden paths that are interleaved with a core of length at least 2. Assume for a contradiction that any such walk W could contain more than one non-cut arc. The non-cut arcs cannot be outside of the core, since all non-core arcs are the only outgoing or the only incoming arcs of some node. If there are at least two non-cut arcs in the core, we can construct an arc-covering closed walk W' that does not contain W. Start with any arc-covering closed walk and repeat it twice. In the first repetition, replace any occurrence of the first non-cut arc of W with a walk that avoids the non-cut arc. And in the second repetition, replace any occurrence of the last non-cut arc of W with a walk that avoids the non-cut arc. Such avoiding walks exist since the avoided arcs are not cut arcs. Further, by avoiding an arc of W, they do not have W as subwalk. The resulting walk W' is arc-covering and closed, but does not have W as subwalk. Hence, W has a forbidden path by Lemma 19. Finally, each walk without forbidden path that is interleaved with a core of length at least 2 produces at most two interleaved safe walks. Since non-interleaved walks without forbidden path cannot be broken into interleaved ones, there are at most $O(n)$ interleaved maximal safe walks. ◀