Beyond the BEST Theorem: Fast Assessment of Eulerian Trails

Alessio Conte, Roberto Grossi, Grigorios Loukides, Nadia Pisanti, Solon Pissis, Giulia Punzi

Beyond the BEST Theorem: Fast Assessment of Eulerian Trails

Conference Paper · September 2021

CITATIONS
0

READS
42

6 authors, including:

Alessio Conte
Università di Pisa
50 PUBLICATIONS **201** CITATIONS
SEE PROFILE

Roberto Grossi
Università di Pisa
188 PUBLICATIONS **3,894** CITATIONS
SEE PROFILE

Grigorios Loukides
King's College London
94 PUBLICATIONS **1,319** CITATIONS
SEE PROFILE

Nadia Pisanti
Università di Pisa
95 PUBLICATIONS **1,111** CITATIONS
SEE PROFILE

Beyond the BEST Theorem: Fast Assessment of Eulerian Trails

Alessio Conte[1][0000−0003−0770−2235], Roberto Grossi[1,2][0000−0002−7985−4222], Grigorios Loukides[3][0000−0003−0888−5061], Nadia Pisanti[1,2][0000−0003−3915−7665], Solon P. Pissis[2,4,5][0000−0002−1445−1932], and Giulia Punzi[1][0000−0001−8738−1595]

[1] Università di Pisa, Pisa, Italy
{conte,grossi,pisanti}@di.unipi.it, giulia.punzi@phd.unipi.it
[2] ERABLE Team, Lyon, France
[3] King's College London, London, UK grigorios.loukides@kcl.ac.uk
[4] CWI, Amsterdam, The Netherlands solon.pissis@cwi.nl
[5] Vrije Universiteit, Amsterdam, The Netherlands

Abstract. Given a directed multigraph $G = (V, E)$, with $|V| = n$ nodes and $|E| = m$ edges, and an integer z, we are asked to assess whether the number $\#ET(G)$ of node-distinct Eulerian trails of G is at least z; two trails are called *node-distinct* if their *node* sequences are different. This problem has been formalized by Bernardini et al. [ALENEX 2020] as it is the core computational problem in several string processing applications. It can be solved in $\mathcal{O}(n^\omega)$ arithmetic operations by applying the well-known BEST theorem, where $\omega < 2.373$ denotes the matrix multiplication exponent. The algorithmic challenge is: *Can we solve this problem faster for certain values of m and z?* Namely, we want to design a combinatorial algorithm for assessing whether $\#ET(G) \geq z$, which does not resort to the BEST theorem and has a predictably bounded cost as a function of m and z. We address this challenge here by providing a combinatorial algorithm requiring $\mathcal{O}(m \cdot \min\{z, \#ET(G)\})$ time.

1 Introduction

Eulerian trails (or Eulerian paths) were introduced by Euler in 1736: Given a multigraph $G = (V, E)$, an Eulerian trail traverses every edge in E exactly once, allowing for revisiting nodes in V. An Eulerian cycle is an Eulerian trail that starts and ends on the same node in V. The perhaps most fundamental algorithmic question related to Eulerian trails is whether we can efficiently identify one of them. Hierholzer's paper ([9],[5, 1B]) can be employed for this purpose to get a linear-time algorithm. A related question is *counting* Eulerian trails: this is $\#P$-complete for undirected graphs [6], while for directed graphs the number of Eulerian trails can be computed in polynomial time using the BEST theorem [1], named after de Bruijn, van Aardenne-Ehrenfest, Smith and Tutte.

Two trails are called *node-distinct* if their *node* sequences are different. Bernardini et al. formalized the following basic problem in [3], which surprisingly had not been previously posed to the best of our knowledge: Given a directed multigraph $G = (V, E)$, with $|V| = n$ nodes and $|E| = m$ edges, two nodes $s, t \in V$,

and a positive integer z, assess whether the number $\#ET(G)$ of node-distinct Eulerian trails of G with source s and target t is at least z. This is the core computational problem in several string processing applications [11, 12, 3].

This problem can be solved in $\mathcal{O}(n^\omega)$ arithmetic operations as follows [3, 11], where $\omega < 2.373$ denotes the matrix multiplication exponent [16, 8, 2]. (The underlying assumption is that G is Eulerian[6], that is, the indegree equals the outdegree in each node, possibly except for the source and the target of the trail.) Let $A = (a_{uv})$ be the adjacency matrix of G allowing both $a_{uv} > 1$ (multi-edges) and $a_{uu} > 0$ (self-loops). Let $r_u = d^+(u)$ for $u \neq t$, $r_t = d^+(t) + 1$, where $d^+(u)$ denotes the outdegree of u, and the edges are counted with multiplicity. We can apply the BEST theorem using its formulation for directed multigraphs [10]:

$$\#ET(G) = (\det L) \cdot \left(\prod_{u \in V} (r_u - 1)! \right) \cdot \left(\prod_{(u,v) \in E} (a_{uv})! \right)^{-1} \tag{1}$$

where $L = (l_{uv})$ is the $n \times n$ matrix with $l_{uu} = r_u - a_{uu}$ and $l_{uv} = -a_{uv}$. The original BEST theorem states that the number of Eulerian trails of a (directed) graph can be obtained by multiplying the number of arborescences rooted at any node of the graph (given by $\det L$) by the number of permutations of the edges outgoing from each node ($\prod_{u \in V}(r_u - 1)!$). In the multigraph version given in Equation (1), the formula is further divided by the number of permutations of multi-edges ($\prod_{(u,v) \in E} a_{uv}!$), in order to only count node-distinct trails.

Beyond the BEST Theorem: We address the following algorithmic challenge [3, Final Remarks]: design a combinatorial algorithm for assessing $\#ET(G) \geq z$, which does not resort to the BEST theorem and has a predictably bounded cost as a function of m and z (as we do not need $\#ET(G)$ to provide an answer).

We first illustrate how this challenge is well-founded *without* matrix multiplication. In Eq. (1), we can single out two factors: the determinant $\det L$ and the ratio of factorials $F = \prod_{u \in V}(r_u - 1)!(\prod_{(u,v) \in E} a_{uv}!)^{-1}$. First, the assessment based on Eq. (1) cannot rely on the assumption that $F \geq z$ (which would imply that $\#ET(G) \geq z$), since we can have $F \ll 1$. As an example, consider a directed multi-cycle with n nodes, $u_1, \ldots, u_n$, each connected to the next (and u_n back to u_1) with k multi-edges: we have only one node-distinct Eulerian trail, so $\#ET(G) = 1$, but there are k^{n-1} arborescences, so $\det L = k^{n-1}$. In particular, we have that $F = \frac{k!(k-1)!^{n-1}}{k!^n} = \frac{1}{k^{n-1}} \ll 1$, for any choice of $s = t$. Second, enumerating arborescences [7, 15], progressively bounding $\det L$ to check whether $\det L \geq z/F$, might be costly. This is because the number of arborescences could be exponential in $\#ET(G)$, as in our example. Therefore, in this paper, we follow a fundamentally different approach, which takes $\mathcal{O}(m) = \mathcal{O}(nk)$ time in our example, and can be generalized to more involved graphs.

We remark that exploiting Eq. (1) *with* matrix multiplication could be costly too. Avoiding matrix multiplication makes a difference of several orders of mag-

[6] If G is not Eulerian, the answer is trivially negative for any non-zero z.

nitude as these arithmetic operations can be costly. In typical instances in experiments, computing $\det L$ with state-of-the-art (sparse) matrix multiplication libraries and other tricks can still take several hours (see [3] for more details).

Our Results and Techniques: Our main contribution is to introduce an approach that does not merely employ the structure of the BEST theorem, but it actually goes beyond that: we design an efficient algorithm which directly provides an assessment by looking directly at Eulerian trails, without considering the different factors of Eq. (1). We first present a natural (but non-trivial) algorithm to facilitate the reader's comprehension. The main idea consists in providing a lower bound on $\#ET(G)$, based on the product of the lower bounds for the node-distinct trails of its strongly connected components. This lower bound is then progressively refined by considering any arbitrarily chosen component, and its contribution is improved by employing some novel structural properties of strongly connected components of Eulerian graphs. Our method conceptually provides a recursive enumeration approach whose calls enumerate the first z node-distinct Eulerian trails in $\Theta(m^2 \cdot \min\{z, \#ET(G)\})$ time. However, we improve upon that, as our lower-bound driven algorithm does not necessarily perform all the recursive calls to assess whether or not $\#ET(G) \geq z$.

The above algorithm may require quadratic time per Eulerian trail because each call might require $\mathcal{O}(m)$ time. We refine it to bring its complexity down by a double numbering on the edges, which guarantees that every call generates *at least two distinct calls*. This double numbering gives us insight on the interior connectivity structure of the graph; namely, how strongly connected components change when we start removing edges, which is the source of the quadratic time. With this double numbering, we manage to instantly retrieve edges that generate new trails, no longer needing to iterate for $\mathcal{O}(m)$ unsuccessful steps. We thus reduce the time by a factor of m, which gives a time-optimal algorithm for $z = \mathcal{O}(1)$ or for $\#ET(G) = \mathcal{O}(1)$. Our main result is formalized as follows.[7]

Theorem 1. *Given a directed multigraph $G = (V, E)$, with $|E| = m$, and an integer z, assessing $\#ET(G) \geq z$ can be done in $\mathcal{O}(m \cdot \min\{z, \#ET(G)\})$ time.*

Let us remark that our algorithms can potentially run asymptotically faster than the worst-case bounds given above. Under suitable assumptions and values of z, it is possible that they run in less than Constant Amortized Time (CAT) per solution (cf. [13]). In principle, this implies that, under suitable assumptions, assessment might be intrinsically more efficient than counting.

Paper Organization: Section 2 introduces the basic definitions and notation used throughout. In Section 3, we prove the combinatorial properties of Eulerian graphs which form the basis of our technique. In Section 4, we present the simple $\mathcal{O}(m^2 \cdot \min\{z, \#ET(G)\})$-time algorithm. This algorithm is then refined to our main result, which is described in Section 5.

[7] We assume throughout that basic arithmetic operations take constant time, which is the case when $z = \mathcal{O}(\text{poly}(m))$.

2 Definitions and Notation

Consider a directed graph $G = (V, E)$ with multi-edges and self-loops, and let $|V| = n$ and $|E| = m$; the edges are counted with multiplicity. A *trail* over G is a sequence of adjacent distinct edges. Two trails are *node-distinct* if their *node* sequences are different. An *Eulerian trail* of G is a trail that traverses every edge exactly once. We consider *node-distinct* Eulerian trails. The set of node-distinct Eulerian trails of G is denoted by $ET(G)$ and its size is denoted by $\#ET(G)$. We may omit the term "node-distinct" when it is clear from its context.

Given a node $u \in V$, we define its *outdegree* (resp. *indegree*) as the number of edges of the form (u, v) (resp. (v, u)), counting multiplicity and self-loops. We then denote by $\Delta(u)$ the difference outdegree(u) − indegree(u). Furthermore, we define the set of *out-neighbors* of u as $N^+(u) = \{v \in V \mid (u, v) \in E\}$. Finally, we use the notation $N^+_C(u) = N^+(u) \cap C$, when referring only to the out-neighbors inside some subgraph C of G.

G is called *strongly connected* if there is a trail in each direction between each pair of the graph nodes. A *strongly connected component* (SCC) of G is a strongly connected subgraph of G. G is called *weakly connected* if replacing all of its edges by undirected edges produces a *connected* graph: it has at least one node and there is a trail between every pair of nodes.

Definition 1. *A directed graph $G = (V, E)$ is* Eulerian with source s and target t*, where $s, t \in V$, if it is weakly connected and (i) $\Delta(s) = 1$, $\Delta(t) = -1$, and $\Delta(u) = 0$ for all $u \in V \setminus \{s, t\}$; or (ii) $\Delta(u) = 0$ for all $u \in V$. In Case (i), G has an* Eulerian trail from s to t*. In Case (ii), G has an* Eulerian cycle*: an Eulerian trail that starts and ends on $s = t$.*

3 Structure and Properties of Directed Eulerian Graphs

The SCCs of a directed Eulerian graph G induce a directed acyclic graph G_{SCC}. Considering this graph, we derive some non-trivial and useful properties, upon which we will heavily rely to design our algorithms for assessing the number of node-distinct Eulerian trails. Let us start with the following crucial lemma whose proof is deferred to the full version of the paper.

Lemma 1. *Let G be an Eulerian graph, with SCCs $C_0, \ldots, C_k$, source $s \in C_0$, and target $t \in C_k$. The corresponding G_{SCC} is a* chain graph *of the form $C_0 \to C_1 \to \ldots \to C_k$, where the arrow between C_i and C_{i+1} represents a single edge $(t_i, s_{i+1}) \in E$, called* bridging edge*. Furthermore, each C_i is Eulerian with source s_i and target t_i, where $s_0 = s, t_k = t$.*

It follows from Lemma 1 that every trail from s to t must traverse all edges of $C_0, \ldots, C_i$ before crossing the bridging edge (t_i, s_{i+1}). As a consequence, we obtain the following.

Corollary 1. *Let G be an Eulerian graph with SCCs $C_0, \ldots, C_k$. Then we have that $ET(G) = \prod_{i=0}^{k} ET(C_i)$, where $\prod$ denotes the cartesian product. It follows that the number of trails of G is the product of the number of trails of its SCCs.*

We can thus focus on an individual SCC or, equivalently, assume, wlog, that the Eulerian graph is strongly connected. The following lemma, whose proof is deferred to the full version of the paper, forms the basis of our technique.

Lemma 2. *Let C be a strongly connected Eulerian graph with source s and target t. For every edge (s, u), there is an Eulerian trail of C whose first two traversed nodes are s and u. Moreover, the residual graph $C \setminus (s, u)$ remains Eulerian with new source u.*

Corollary 2. *Let C_i be any SCC of an Eulerian graph with source s_i. Then:*

$$ET(C_i) = \bigcup_{u \in N^+_{C_i}(s_i)} (s_i, u) \cdot ET(C_i \setminus (s_i, u)),$$

that is, the Eulerian trails of C_i are given by concatenating each possible start of the trail (s_i, u) with all its possible continuations, i.e., the trails in $ET(C_i \setminus (s_i, u))$. Thus the number of trails of C_i is the sum of the number of trails of the subgraphs with edges (s_i, u) removed, for every $u \in N^+_{C_i}(s_i)$ distinct out-neighbor of s_i in C_i, with u as the new source.

Proof. This follows from Lemma 2 applied to the SCCs: we know that each distinct out-neighbor of s_i leads to at least one trail; furthermore, no two of these trails can be equal since they begin with distinct edges. Lastly, all trails are accounted for, since we consider every trail starting from every distinct out-neighbor of s_i, and s_i is the source of C_i. □

Note the subtle point in the statement of Corollary 2, where we use $N^+_{C_i}(s_i)$ instead of $N^+(s_i)$: if the latter two differ, it is because s_i has an outgoing bridging edge, and this should be traversed *after* all other edges in C_i.

4 Assessment Algorithm for #ET(G)

We present AssessET, a simple but non-trivial algorithm for assessing the number of node-distinct Eulerian trails on a given directed graph, which will be refined in Section 5. AssessET takes the following input parameters: (i) a weakly connected Eulerian graph $G = (V, E)$ with source s and target t; (ii) a positive integer threshold z, and (iii) a function $lb(\cdot)$, which outputs a lower bound on the number of the node-distinct Eulerian trails in G. To achieve the desired complexity, $lb(\cdot)$ must be computable in $\mathcal{O}(m)$ time and $lb(\cdot) \geq 1$ must hold.

Proposition 1. *Given graph G, nodes s and t, integer z, and $lb(\cdot)$,* AssessET *assesses $\#ET(G) \geq z$ in $\mathcal{O}(m^2 \cdot \min\{z, \#ET(G)\})$ time using $\mathcal{O}(mz)$ space.*

Main Idea: Let $C_0, \ldots, C_k$ be the set of SCCs of an Eulerian graph G as illustrated in Lemma 1. AssessET exploits Corollary 1, Lemma 2, and Corollary 2, to provide a lower bound on the number of node-distinct Eulerian trails of graph G, denoted by $lb_{ET}(G)$, where $lb_{ET}(G) \leq \#ET(G)$. Initially, we set

$lb_{ET}(G) = \prod_{i=0}^{k} lb(C_i)$, based on the product of the lower bounds for the number of node-distinct Eulerian trails of the SCCs of G by Corollary 1. Then $lb_{ET}(G)$ is progressively refined by considering any arbitrarily chosen component, say C_i, and in turn replacing its lower bound $lb(C_i)$ with a new lower bound $lb_{ET}(C_i)$ that exploits Lemma 2 and its Corollary 2. That is, we remove each different outgoing edge from the source s_i of C_i, and after computing the $lb(\cdot)$ function on all of the resulting graphs, we sum these lower bounds to obtain $lb_{ET}(C_i)$, and update $lb_{ET}(G)$. We proceed in this way until either $lb_{ET}(G) \geq z$, or we compute the actual number of trails: $lb_{ET}(G) = \#ET(G)$.

The requirements for the lower bound function are trivially satisfied by the constant function $lb(\cdot) \equiv 1$. However, we use a better lower bound given by Lemma 3 below, whose proof is deferred to the full version of the paper.

Lemma 3. *For any Eulerian graph G, the function*

$$lb(G) = 1 + \sum_{v \in V(G) : |N_G^+(v)| \geq 3} (|N_G^+(v)| - 2). \tag{2}$$

is a lower bound for the number $\#ET(G)$ of node-distinct Eulerian trails of G.

Function ComputeSCC*:* Our algorithm relies on a function ComputeSCC(G), which computes the SCCs of a given input graph G. This function only outputs the non-trivial components (i.e., comprised of multiple nodes), and it requires $\mathcal{O}(m)$ time to achieve this (specifically, we make use of [14]).

Frontier Data Structure: In order to efficiently explore the different SCCs as discussed above, we introduce the *Frontier Data Structure*, denoted by $\mathcal{F} = \{f_1, \ldots, f_{|\mathcal{F}|}\}$, representing the frontier of the recursive tree we are *implicitly* constructing when traversing a component. At any moment of the computation, an element $f_j \in \mathcal{F}$ is a tuple $\langle C_0^j, \ldots, C_{h_j}^j \rangle$, where $C_0, \ldots, C_{h_j}$ are the non-trivial SCCs of some Eulerian subgraph $G_j \subset G$. A component is considered *trivial* if it is comprised of a single node. A trivial component is omitted because it contributes to the product in Equation (3) below by a factor of one. Different G_j's are obtained from G by removing different edges that are outgoing from the source of a component, as per Lemma 2; thus G_j differs from any other G_l by at least one removed edge. In this way, each element of the frontier represents at least one node-distinct Eulerian trail of G. Furthermore, our data structure $\mathcal{F}$ retains an important invariant: at any moment, the elements of $\mathcal{F}$ are the SCC decompositions of the subgraphs which realize the current bound. That is,

$$lb_{ET}(G) = \sum_{j=1}^{|\mathcal{F}|} lb_{ET}(f_j) = \sum_{j=1}^{|\mathcal{F}|} \prod_{i=0}^{h_j} lb(C_i^j). \tag{3}$$

Each component $f_j[i] = C_i^j$, with source s_i^j and target t_i^j, is represented in $f \in \mathcal{F}$ as a tuple of the form $(V[C_i^j], E[C_i^j], s_i^j, t_i^j, lb(C_i^j))$. In what follows, we consider $\mathcal{F}$ implemented as a *stack*: both removing and inserting elements

requires $\mathcal{O}(1)$ time with POP and PUSH operations. Performing these operations also modifies the size of $\mathcal{F}$, which is accounted for. We can thus answer whether the stack is empty in $\mathcal{O}(1)$ time.

Algorithm AssessET*:* The algorithm maintains a running bound lb_{ET}, induced by the components currently forming the elements of the stack, according to Equation (3), where lb_{ET} is the current value of $lb_{ET}(G)$. We proceed as follows:

1. Compute the (non-trivial) SCCs of graph G. If there is none, we only have one trail, and $lb_{ET} = 1$. Otherwise, we initialize the stack with the tuple $\langle C_0, \ldots, C_k \rangle$ of these SCCs, and also initialize the bound accordingly setting $lb_{ET} \leftarrow \prod_{j=0}^{k} lb(C_j)$.
2. While $lb_{ET} < z$, we perform the following:
   (a) If the stack is empty, we output NO. Since non-trivial components are never added into the stack, the stack is empty if and only if $lb_{ET} = \#ET(G)$ and $lb_{ET} < z$.
   (b) Otherwise, we pop an element f from the stack, and remove its contribution from the current bound: $lb_{ET} \leftarrow lb_{ET} - lb_{ET}(f)$, where $lb_{ET}(f) = \prod_{i=1}^{|f|} lb(f[i])$.
   (c) We pick an arbitrary component $C_i = f[i]$ of tuple f, and let s_i be its source. We remove the component from f.
   (d) For all distinct out-neighbors $u \in N^+_{C_i}(s_i)$:
      i. We compute the SCCs $\mathcal{C}$ of C_i with edge (s_i, u) removed.
      ii. If f with the added new components $\mathcal{C}$ (i.e. $f \cdot \mathcal{C}$) is non-empty, we add it into the stack and increase the running bound accordingly as $lb_{ET} \leftarrow lb_{ET} + lb_{ET}(f \cdot \mathcal{C})$. If $f \cdot \mathcal{C}$ is empty, it corresponds to a single Eulerian trail, so we increase the bound lb_{ET} by one.
3. If we exit from the while loop in Step 2, then $lb_{ET} \geq z$ and we output YES.

When $lb(\cdot)$ always returns 1, AssessET makes $\mathcal{O}(mz)$ calls to compute the SCCs, of $\mathcal{O}(m)$ time each, as it essentially enumerates z Eulerian trails one by one. However, when $lb(\cdot) > 1$, a lot of these calls are avoided as lower bounds are multiplied. The pseudocode of AssessET is provided in Algorithm 1.

The correctness of AssessET follows from Corollary 1, Lemma 2 and Corollary 2. The analysis of time and space complexity of AssessET, which completes the proof of Proposition 1, is deferred to the full version of the paper.

5 Improved Assessment Algorithm

We may think of AssessET as a recursive computation (handled explicitly with pop/push on a stack) having the drawback that it makes $\mathcal{O}(mz)$ recursive calls. To try and speed up the process, one could resort to existing decremental SCC algorithms [4]. However, these tend to add (poly)logarithmic factors, and do not immediately yield improvements unless further amortization is suitably designed.

We use a different approach, reducing the number of calls to $\mathcal{O}(z)$ by guaranteeing that each call generates at least two further calls or immediately halts

Algorithm 1 (AssessET)

```
1: procedure AssessET(G = (V, E), z = O(poly(|E|), lb(·))
2:     C_0, ..., C_k ← ComputeSCC(G)              ▷ Only considers non-trivial SCCs
3:     f ← ⟨C_0, ..., C_k⟩
4:     if f is empty then lb_ET ← 1
5:     else Stack.push(f)                                          ▷ Initialization
6:         lb_ET ← ∏_{j=0}^{k} lb(C_j)
7:     while lb_ET < z do
8:         if Stack.IsEmpty() then Output NO
9:         f ← Stack.pop()
10:        lb_ET ← lb_ET − lb_ET(f)                    ▷ lb_ET(f) = ∏_{i=1}^{|f|} lb(f[i])
11:        Choose any i; let C_i = f[i] and s_i be its source   ▷ f[i] is the i-th SCC of f
12:        Remove C_i from f
13:        for all u ∈ N+_{C_i}(s_i) do
14:            C ← ComputeSCC(C_i \ (s_i, u))
15:            if f · C is not empty then      ▷ f · C: f with each SCC of C appended
16:                Stack.push(f · C)
17:                lb_ET ← lb_ET + lb_ET(f · C)
18:            else lb_ET ← lb_ET + 1
19:    Output YES
```

when one Eulerian trail is found. In this section, we show how to attain this goal with an efficient combinatorial procedure.

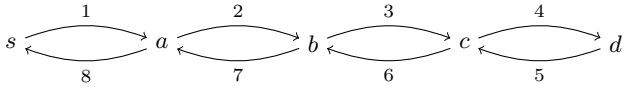

Fig. 1: This graph has a single node-distinct Eulerian trail from s to itself, even though all nodes except for s and d are branching.

5.1 Introducing Function BranchingSource

Consider the SCC C_i chosen in AssessET, and its source s_i. We call a node $u \in C_i$ *branching* if it has at least two distinct out-neighbors in C_i, that is, $|N^+_{C_i}(u)| \geq 2$. Thus, if s_i is branching, we have at least two calls by Lemma 2. The issue comes when s_i has just one out-neighbor, as illustrated in Figure 1: some of the remaining nodes could be branching but, unfortunately, only one node-distinct Eulerian trail exists. Thus, the existence of branching nodes when the source s_i is not branching does not guarantee that we attain our goal.

One first solution comes to mind, as it is exploited in our lower bound $lb(\cdot)$ of Equation 2. Consider a trail $T \in ET(C_i)$, which is nonempty as C_i is Eulerian: a node u gives rise to at least $|N^+_{C_i}(u)| - 2$ further Eulerian trails by Lemma 2

as, when u becomes a source for the first time, one out-neighbor of u is part of T and at most one out-neighbor of u leads to a bridging edge; thus the remaining $|N^+_{C_i}(u)| - 2$ out-neighbors can be traversed in any order by so many other Eulerian trails. While this helps for $|N^+_{C_i}(u)| \geq 3$, it is not so useful in the situation illustrated in Figure 1, where all branching nodes have $|N^+_{C_i}(u)| = 2$.

Main Idea: A better solution is obtained by introducing a function BranchingSource to be applied to any tuple f of SCCs from the frontier data structure $\mathcal{F}$. If any of these SCCs has a branching source, then BranchingSource returns f itself. Otherwise, it examines each SCC C in f: if $\#ET(C) = 1$, it removes C from f as it is trivial; otherwise, it finds the longest common prefix P of all trails in $ET(C)$, and computes the SCCs of $C \setminus P$, which take the place of C in f. Among these SCCs, one is guaranteed to have a branching source, so BranchingSource returns f updated in this way. Note that only trivial SCCs are removed by BranchingSource, and hence the number of Eulerian trails cannot change. If f is empty, then we have a single Eulerian trail as there is no choice. BranchingSource can be implemented in $\mathcal{O}(m^2)$ time as it simulates what AssessET does until a branching source is found. The challenge is to implement it in $\mathcal{O}(m)$ time. Armed with that, we can modify AssessET and get ImprovedAssessET, where we guarantee in $\mathcal{O}(m)$ time that *the source s_i is always branching*. The modification is just a few lines, once BranchingSource is available, so we do not provide a detailed description of the pseudocode.

5.2 Linear-time Computation of BranchingSource

Suppose that tuple f in the frontier data structure $\mathcal{F}$ contains only SCCs with non-branching sources (otherwise, BranchingSource returns f unchanged). Consider any SCC C in the tuple f. The main idea is to fix any trail $T \in ET(C)$, which can be found in $\mathcal{O}(|E(C)|)$ time, and traverse T asking at each node u whether there is an alternative trail T' branching at u.

Swap Edges: Let us start with the following definition.

Definition 2. *Given an Eulerian trail T of an SCC C, let T_u be the prefix of T from its source s to the first time u is met, and let (u, v) be the next edge traversed by T. An edge (u, v') in C is a* swap edge *if $T_u \cdot (u, v')$ is prefix of another Eulerian trail $T' \neq T$ and $v' \neq v$. We say that u admits a swap edge and $T_u = T'_u$ is* the longest common prefix *of T and T'.*

The discovery of swap edges in C is key to BranchingSource: although different Eulerian trails of C may give rise to different swap edges in C, these trails all share P, so the node u at the end of P can be identified by T_u (Definition 2), for *any* trail $T \in ET(C)$. The proof of the following lemma is deferred to the full version of the paper.

Lemma 4. *Suppose that all swap edges are known in an SCC C of f for any given trail $T \in ET(C)$. Then (i) $\#ET(C) = 1$ if and only if there are no swap*

edges in C; moreover, (ii) if $\#ET(C) > 1$, let u be the first node that is met traversing T and that admits a swap edge. Then $P = T_u$ is the longest common prefix of all the trails in $ET(C)$.

Using Swap Edges in BranchingSource: Based on Lemma 4, BranchingSource examines each $C \in f$: it tests whether C is trivial ($\#ET(C) = 1$), or it finds the longest common prefix $P = T_u$ of all the trails. If all SCCs are trivial, it returns an empty f. Otherwise, it deletes from f the trivial SCCs found so far, and for the current non-trivial SCC C, it computes the set $\mathcal{C} = \text{ComputeSCC}(C \setminus T_u)$ of SCCs. Note that u becomes the source of an SCC in $\mathcal{C}$ and u is branching as it admits a swap edge (u, v'), along with (u, v) from its trail T. Thus, u keeps at least two out-neighbors v and v' in $\mathcal{C}$. At this point, BranchingSource stops its computation, updates f by replacing C with the SCCs from $\mathcal{C}$, and returns f. Since only trivial SCCs are removed from f, and the number of Eulerian trails in C is the product of those in the SCCs of $\mathcal{C}$, the overall number of Eulerian trails in f does not change before and after its update. This proves the following.

Lemma 5. *Given any tuple f in $\mathcal{F}$ and the set of swap edges in the SCCs of f, the function* BranchingSource *takes $\mathcal{O}(m)$ time to update f, so that either f is empty (a single Eulerian trail exists in f), or f contains at least one SCC with branching source.*

Remark 1. Since every swap edge generates *at least one new Eulerian trail*, if we can find all swap edges in $\mathcal{O}(m)$ time, we can employ $lb(G) = 1+$ "number of swap edges" in our algorithm. Any node with three different out-neighbors generates at least a swap edge. Thus, this new choice for the lower bound function necessarily performs better than the one shown in Equation (2). For example, in Figure 2, there are 5 swap edges whereas $lb(\cdot) = 1$.

Finding Swap Edges in Linear Time: We are thus interested in finding all the swap edges in linear time. We need the following property, whose proof is deferred to the full version of the paper, to characterize them for an SCC C of f.

Lemma 6. *Let C be an SCC, and let T with prefix $T_u \cdot (u, v)$ be one of its Eulerian trails. Edge (u, v'), for $v' \neq v$, is a swap edge if and only if there is a trail from v' to u (i.e., u is reachable from v') in $C \setminus T_u$.*

In order to find the swap edges, we need to traverse T in reverse order and assign each edge $e \in E(C)$ two integers, as illustrated in the example of Figure 2: (i) the *Eulerian trail numbering* $etn(e)$, which represents the position of e inside T and is immediate to compute, and (ii) the *disconnecting index* $di(e)$, which is discussed in the next paragraph as its computation is a bit more involved. As we will see (Lemma 7), comparing these integers allows us to check if a given edge is a swap edge in constant time.

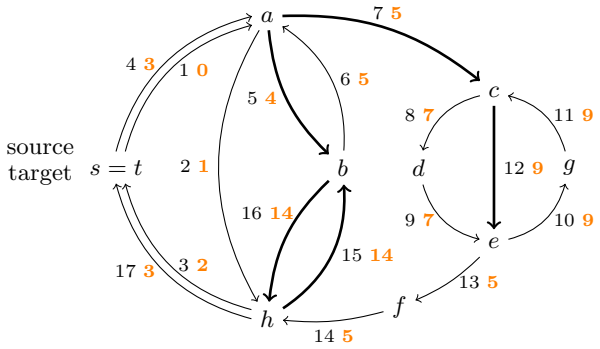

Fig. 2: Example of an Eulerian graph with $s = t$. The black (left) numbers on the edges are the Eulerian trail numbers *etn* for a given trail T; the orange (right) ones are the disconnecting indices *di*s. Swap edges are in bold.

Disconnecting Indices: We introduce the notion of disconnecting index relatively to a given trail $T \in ET(C)$, according to the following rationale. We observe that Lemma 5 characterizes a swap edge (u, v') by stating that u and v' must belong to the same SCC after T_u is removed from C. Suppose that we want to traverse T to discover the swap edges. Equivalently, we take the edges according to their *etn* order in T. Fix any edge (u, v'). At the beginning, u and v' are in the same SCC C. Next, we start to conceptually remove, from C, the edges traversed by an increasingly long prefix of T: how long will u and v' stay in the same SCC? In this scenario, the disconnecting index of (u, v') corresponds to the maximum *etn* (hence prefix of T) for which u and v' will stay in the same SCC, i.e., removing any prefix of T longer than this one from C disconnects v' from u.

For any $\ell \in [0, m]$, we denote by $T_{\leq \ell}$ the prefix T_u of T such that $|T_u| = \ell$. When $\ell = 0$, it is the empty prefix; when $\ell = m$, it is T itself.

Definition 3. *Given an edge $(u, v') \in E(C)$, its* disconnecting index *is*

$$di(u, v') = \max\{0 \leq \ell < etn(u, v') \mid u, v' \text{ are inside an SCC of } C \setminus T_{\leq \ell}\}.$$

Figure 2 illustrates an example where the following property, whose proof is deferred to the full version of the paper, can be checked by inspection.

Lemma 7. *For any edge $(u, v') \in E(C)$, we have that (u, v') is a swap edge for a given trail if and only if $di(u, v') \geq etn(u, v) - 1$ for some $v \neq v'$.*

Linear-Time Computation of Disconnecting Indices: Consider an SCC C from $f \in \mathcal{F}$, and any arbitrary trail $T \in ET(C)$ (computable in $\mathcal{O}(|E(C)|)$ time). Assign the Eulerian trail numbering $etn(e)$ to each edge $e \in E(C)$. We discuss how to assign the disconnecting index $di(e)$ to each edge e in $\mathcal{O}(|E(C)|)$ time.

We proceed by reconstructing T backwards. That is, we conceptually start from an empty graph, and we add edges from T, one at a time from last to first

(i.e., in decreasing order of their *etn* values), until all edges from T are added back obtaining again the SCC C. During this task, along with disconnecting indices, we also assign a *flag* $tr(u) = true$ to the nodes u touched by the edges that have been added. We keep a stack, BRIDGES, for the edges that have been added but do not yet have a disconnecting index, i.e., they are not in an SCC of the current partial graph. More formally, we will guarantee the following invariants:

I1 The edges in BRIDGES have increasing *etn* values, starting from the top.

I2 The edges in BRIDGES are all and only the bridging edges of the current graph.

I3 Given any two consecutive edges e, e' in BRIDGES, the edges with *etn* values in $[etn(e) + 1, etn(e') - 1]$ (which, observe, are not in BRIDGES) make up an SCC of the current graph.

I4 The flag $tr(u)$ is *true* if and only if u is incident to an edge of the current graph.

We describe the algorithm, prove its correctness and all invariants.

For $\ell = m, m-1, \ldots, 1$, step ℓ adds back to the current graph the edge (u, v) such that $etn(u, v) = \ell$. Let u be the tail and v be the head of the edge.

- If the tail u has not been explored yet (i.e., $tr(u) = false$), we add (u, v) to BRIDGES and set $tr(u) = true$. If $\ell = m$, then v is the last node of the trail and we also set $tr(v) = true$.
- Otherwise, u has been traversed before, and there must be at least an edge incoming in u in our current graph; let (z, u) be the one such edge with highest *etn* value, say, $etn(z, u) = x$. We assign $di(u, v) = etn(u, v) - 1$, and pop all edges e from BRIDGES such that $etn(e) \leq x$, assigning $di(e) = etn(u, v) - 1$ to all of these too.

Lemma 8. *Given an SCC C from f in $\mathcal{F}$, and any arbitrary trail $T \in ET(C)$, the disconnecting indices of T can be computed in $\mathcal{O}(|E(C)|)$ time and space.*

The proof of Lemma 8 is deferred to the full version of the paper. We thus arrive at our main result.

Theorem 1. *Given a directed multigraph $G = (V, E)$, with $|E| = m$, and an integer z, assessing $\#ET(G) \geq z$ can be done in $\mathcal{O}(m \cdot \min\{z, \#ET(G)\})$ time.*

Other than being easily extensible to the *edge-distinct* case, the algorithm underlying Theorem 1 has an attractive property: its number of $\mathcal{O}(m)$-time steps is z in the worst case, but can be significantly smaller in practice, thanks to suitable lower bounding techniques. This property means that our assessment algorithm can potentially run in less than CAT per solution on favorable instances.

Acknowledgments: We wish to thank Luca Versari for useful discussions on a previous version of the main algorithm. This paper is part of the PANGAIA project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 872539. This paper is also part of the ALPACA project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 956229.

References

1. van Aardenne-Ehrenfest, T., de Bruijn, N.G.: Circuits and Trees in Oriented Linear Graphs, pp. 149–163. Birkhäuser Boston, Boston, MA (1987). https://doi.org/10.1007/978-0-8176-4842-8_12
2. Alman, J., Williams, V.V.: A refined laser method and faster matrix multiplication. In: Marx, D. (ed.) Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021. pp. 522–539. SIAM (2021). https://doi.org/10.1137/1.9781611976465.32, https://doi.org/10.1137/1.9781611976465.32
3. Bernardini, G., Chen, H., Fici, G., Loukides, G., Pissis, S.P.: Reverse-safe data structures for text indexing. In: Blelloch, G.E., Finocchi, I. (eds.) Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2020, Salt Lake City, UT, USA, January 5-6, 2020. pp. 199–213. SIAM (2020). https://doi.org/10.1137/1.9781611976007.16, https://doi.org/10.1137/1.9781611976007.16
4. Bernstein, A., Probst, M., Wulff-Nilsen, C.: Decremental strongly-connected components and single-source reachability in near-linear time. In: Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing. p. 365–376. STOC 2019, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3313276.3316335, https://doi.org/10.1145/3313276.3316335
5. Biggs, N.L., Lloyd, E.K., Wilson, R.J.: Graph Theory 1736-1936. Clarendon Press (1976)
6. Brightwell, G.R., Winkler, P.: Counting Eulerian circuits is #P-complete. In: Demetrescu, C., Sedgewick, R., Tamassia, R. (eds.) Proceedings of the Seventh Workshop on Algorithm Engineering and Experiments and the Second Workshop on Analytic Algorithmics and Combinatorics, ALENEX /ANALCO 2005, Vancouver, BC, Canada, 22 January 2005. pp. 259–262. SIAM (2005), http://www.siam.org/meetings/analco05/papers/09grbrightwell.pdf
7. Gabow, H.N., Myers, E.W.: Finding all spanning trees of directed and undirected graphs. SIAM Journal on Computing **7**(3), 280–287 (1978)
8. Gall, F.L.: Powers of tensors and fast matrix multiplication. In: Nabeshima, K., Nagasaka, K., Winkler, F., Szántó, Á. (eds.) International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014. pp. 296–303. ACM (2014). https://doi.org/10.1145/2608628.2608664, https://doi.org/10.1145/2608628.2608664
9. Hierholzer, C., Wiener, C.: Über die möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren. Mathematische Annalen **6**(1), 30–32 (1873)
10. Hutchinson, J.P., Wilf, H.S.: On eulerian circuits and words with prescribed adjacency patterns. Journal of Combinatorial Theory, Series A **18**(1), 80–87 (1975)
11. Kingsford, C., Schatz, M.C., Pop, M.: Assembly complexity of prokaryotic genomes using short reads. BMC Bioinform. **11**, 21 (2010). https://doi.org/10.1186/1471-2105-11-21, https://doi.org/10.1186/1471-2105-11-21
12. Patro, R., Mount, S.M., Kingsford, C.: Sailfish: Alignment-free isoform quantification from RNA-seq reads using lightweight algorithms. Nature Biotechnology **32**, 462–464 (2014). https://doi.org/10.1038/nbt.2862, https://www.nature.com/articles/nbt.2862
13. Ruskey, F.: Combinatorial generation. Preliminary working draft. University of Victoria, Victoria, BC, Canada **11**, 20 (2003)

14. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J. Comput. **1**(2), 146–160 (1972). https://doi.org/10.1137/0201010, https://doi.org/10.1137/0201010
15. Uno, T.: A new approach for speeding up enumeration algorithms and its application for matroid bases. In: International Computing and Combinatorics Conference. pp. 349–359. Springer (1999)
16. Williams, V.V.: Multiplying matrices faster than Coppersmith-Winograd. In: Karloff, H.J., Pitassi, T. (eds.) Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012. pp. 887–898. ACM (2012). https://doi.org/10.1145/2213977.2214056, https://doi.org/10.1145/2213977.2214056