bioRxiv preprint doi: https://doi.org/10.1101/2025.11.21.689636; this version posted November 22, 2025. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

Billi: Provably Accurate and Scalable Bubble Detection in Pangenome Graphs

Shreeharsha G Bhat[1,2,*] Daanish Mahajan[3,*] Chirag Jain[1]

[1]Department of Computational and Data Sciences, Indian Institute of Science, KA 560012, India
[2]Department of Biotechnology, Indian Institute of Technology Madras, TN 600036, India
[3]Department of Interdisciplinary Mathematical Sciences, Indian Institute of Science, KA 560012, India
be21b037@smail.iitm.ac.in, daanishm@iisc.ac.in, chirag@iisc.ac.in

Abstract

A key application of pangenome graphs is the characterization of small and large genomic variants represented as *bubbles* within the graph. Although bubbles have been extensively studied in directed graphs in the context of genome assembly, there remains a need for a rigorous definition and systematic analysis of bubbles in *bidirected* graphs, which is the predominant data structure used to represent pangenomes. We show that existing bubble definitions for bidirected graphs do not fully meet the requirements for representing genetic sites and alleles; in particular, overlapping bubbles may not exhibit strict nesting. To address this, we introduce a new subgraph abstraction called *panbubble* and prove that it satisfies the desired structural properties for variant characterization. We then present an exact algorithm with runtime $\mathcal{O}(|V|^2(|V|+|E|))$ for detecting all panbubbles in a bidirected graph $G=(V,E)$. In addition, we propose a heuristic algorithm that produces identical output as the exact algorithm in practice and scales to large graphs, including both the first and second releases of the Human Pangenome Reference Consortium (HPRC). We implemented our algorithms in the tool `Billi` (github.com/at-cg/billi). On our largest dataset, Billi is more than 15× faster and uses over 5× less memory than VG.

*Should be regarded as joint first-authors.

bioRxiv preprint doi: https://doi.org/10.1101/2025.11.21.689636; this version posted November 22, 2025. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

1 Introduction

Pangenome graphs provide a flexible framework for representing complex structural variations, including cases where smaller variations are nested within larger ones [1, 3, 14, 26, 27]. Unlike conventional variant calling methods, variant discovery in pangenome graphs does not use a single reference genome coordinate system. A key computational challenge in this setting is the identification of *bubbles*, which are substructures in the graph that likely correspond to genomic variation. Informally, a bubble is a region where multiple paths diverge from a common source vertex and subsequently reconverge at a common sink vertex. The concept of bubbles and efficient algorithms for their detection have been extensively studied in the context of genome assembly [2, 11, 23, 28]; however, these studies typically assume a directed graph representation. In contrast, pangenome graph construction tools [10, 12, 19, 20] employ *bidirected graphs* [9] to represent double-stranded DNA.

Previous efforts [8, 20, 22, 24] to generalize the concept of bubbles to bidirected graphs build upon the notion of the *superbubble*, first introduced and formally defined in [23]. Superbubble is an important class of subgraphs in directed graphs, originally motivated by the goal of simplifying genome assembly graphs. This concept enables the identification of substructures within an assembly graph that arise from sequencing errors, heterozygous variants, and near-identical repeats. In their work, Onodera *et al.* [23] precisely defined the conditions under which a pair of distinct vertices s and t can serve as the source and sink of a superbubble, respectively. Specifically, the enclosed superbubble subgraph B must be acyclic; there must be no edge from a vertex outside B to any vertex in $B \setminus \{s\}$; and no edge from a vertex in $B \setminus \{t\}$ to a vertex outside B. Superbubbles exhibit elegant structural properties. For instance, the subgraphs of two superbubbles are either disjoint or strictly nested, and every vertex within a superbubble lies on some source-sink path.

In pangenome graphs, cyclic subgraphs are of particular interest because genomic rearrangements such as inversions and duplications give rise to paths that loop back. Several generalizations of the superbubble concept have been proposed for *bidirected graphs*, including *snarl* [24], *bibubble* [20], and *flubble* [22]. These formulations relax the acyclicity constraint, enabling the representation of more complex graph structures. However, they do not fully preserve all properties of superbubbles. For instance, snarls and flubbles may contain vertices that are not part of any source-sink walk (Supplementary Figure S2). Moreover, it is possible that a pair of snarls, bibubbles, or flubbles in a bidirected graph overlap without being strictly nested (Supplementary Figure S3). This can introduce ambiguity in delineating regions corresponding to genomic variation. Although overlapping subgraphs can be filtered using some heuristic to enforce disjointness, this approach risks missing genuine variation.

In this paper, we extend the superbubble framework [23] to bidirected graphs while preserving key structural properties and accommodating cyclic subgraphs. Our main contributions are as follows:

- We introduce a subgraph abstraction called *panbubble*. This is inspired by the superbubble concept but adapted to bidirected graphs. We also define *hairpin*, a related substructure that corresponds to inversions [22].
- We give formal proofs showing that panbubbles and hairpins exhibit the intended nesting behavior.
- We present an $\mathcal{O}(|V|^2(|V| + |E|))$-time algorithm for finding all panbubbles and hairpins. We also propose a faster $\mathcal{O}(|V|(|V| + |E|))$-time heuristic algorithm that yields identical results as our exact algorithm on the test datasets. We implemented both algorithms in our tool `Billi`.
- `Billi` processes the largest publicly available human pangenome graph, containing over a hundred million vertices, in under 75 minutes and with 90 GB of memory. In comparison, computing snarls using VG on this graph required approximately 20 hours and 500 GB of memory.
- We compare the identified panbubbles with snarls and bibubbles on several pangenome graphs, and observe strong overall agreement. Through Bandage [30] visualizations, we show that the observed differences arise because prior bubble definitions do not comply with all conditions of panbubble.

bioRxiv preprint doi: https://doi.org/10.1101/2025.11.21.689636; this version posted November 22, 2025. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

2 Notations and problem statement

Biedged graph. *Bidirected graphs* are widely used in genomics because they offer a convenient way to represent adjacency relationships between DNA sequences, while allowing traversal of a sequence in both forward and reverse orientations. In this study, we adopt the *biedged graph* representation of bidirected graphs, following the approach of previous works [20, 24]. A biedged graph $G_b = (V_b, E_b, f_b)$ is an undirected, colored multigraph subject to the following three constraints.

- Function $f_b : E_b \to \{\texttt{gray}, \texttt{black}\}$ assigns either gray or black color to each edge.
- Every vertex of the graph is incident with exactly one black edge.
- Between any two vertices (which may be identical), there can be at most one gray edge.

We call two vertices $u, v \in V_b$ $(u \neq v)$ *adjacent* if they are connected by at least one edge of any color. With the above definition, observe that adjacent vertices u and v can have either (i) one gray edge, (ii) one black edge, or (iii) one black and one gray edge between them (Figures 1 and 2A). For every vertex $v \in V_b$, we use $\hat{v}$ to denote the neighbor vertex of v connected using a black edge. We denote the black edge incident to v as e_v. Clearly, $e_v = e_{\hat{v}}$ for all $v \in V_b$. In genome sequence analysis applications where the biedged (or equivalently, bidirected) graph representation is used, each black edge is typically annotated with a DNA sequence. In any walk that traverses a black edge e_v, the order in which vertices v and $\hat{v}$ are visited determines the orientation (forward or reverse complement) in which the sequence label of e_v is to be read. In this work, we omit these sequence labels and focus solely on the underlying graph topology.

The notion of a walk in a biedged graph deviates from the standard definition of walks in undirected graphs. A sequence of vertices $(v_0, v_1, \ldots, v_n)$ with $n \geq 1$ is a *walk* in a biedged graph if (a) for all $1 \leq i \leq n$, there exists an edge between v_{i-1} and v_i, (b) the walk starts and ends using black edges, that is, $v_1 = \hat{v}_0$ and $v_n = \hat{v}_{n-1}$, and (c) the entire walk alternates between black and gray edges. For example, vertex sequence $(v_0, v_1, v_2, v_3, v_4, v_5)$ in Figure 1 forms a valid walk. On the other hand, the vertex sequence $(v_0, v_1, v_6, v_2, v_3)$ is not a walk because it contains two consecutive gray edges. Note that if vertex v appears in a walk, then its neighbor vertex $\hat{v}$ must also appear in that walk. Due to symmetry, $(v_n, v_{n-1}, \ldots, v_1, v_0)$ is a walk if $(v_0, v_1, \ldots, v_{n-1}, v_n)$ is a walk.

Considering any $u, v \in V_b$, we say that v is *reachable* from u if v appears in some walk starting from u. Note that the set of vertices reachable from u always includes u and $\hat{u}$ because $(u, \hat{u})$ is a valid walk. We say that a walk $(v_0, v_1, \ldots, v_{n-1}, v_n)$ passes through v_i for all $1 \leq i \leq n-1$. Given any three vertices $u, v, w \in V_b$, v is said to be reachable from u without passing through w if v appears in some walk starting from u and that walk does not pass through w. We call a vertex not incident with a gray edge a *tip*.

Two vertex-disjoint subgraphs of a biedged graph are said to be *disconnected* in the usual sense, that is, if there exists neither black edge nor gray edge that has one endpoint in the first subgraph and the other endpoint in the second subgraph. Unless a biedged graph has two or more disconnected subgraphs, it is said to be *connected*. A walk $(v_0, v_1, \ldots, v_{n-1}, v_n)$ with $n \geq 3$ is called a *closed walk* if $v_{n-1} = v_0$ and $v_n = v_1$. Similarly, a walk $(v_0, v_1, \ldots, v_{n-1}, v_n)$ with $n \geq 3$ is an *inverted closed walk* if $v_{n-1} = v_1$ and $v_n = v_0$. For example, in Figure 1, the vertex sequence $(v_8, v_9, v_{12}, v_{13}, v_{11}, v_{10}, v_8, v_9)$ is a closed walk, whereas $(v_2, v_3, v_4, v_5, v_3, v_2)$ is an inverted closed walk.

Compact biedged graph. The process of compacting long non-branching paths into single vertices is a common data reduction step in graph-based sequence analysis. In the context of a biedged graph $G_b = (V_b, E_b, f_b)$, a *compaction operation* is defined as follows. Consider two vertices $u, v \in V_b$ with $u \neq v$ and $u \neq \hat{v}$. u and v are said to be *compactable* if (i) they are connected to each other by a gray edge, (ii) the number of gray edges incident to u is one, and (iii) the number of gray edges incident to v is one. The compaction operation on a compactable vertex pair (u, v) adds a black edge between $\hat{u}$ and $\hat{v}$, and then removes u and v from the graph along with all the edges incident to u or v. The resulting graph is a biedged graph with two fewer edges and two fewer vertices. A compact biedged graph is obtained by applying compaction operations as many times as possible in any order on the given biedged graph.

We next introduce the concepts of *hairpin* and *panbubble*, which will be useful to characterize genomic variation in pangenome graphs.

bioRxiv preprint doi: https://doi.org/10.1101/2025.11.21.689636; this version posted November 22, 2025. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

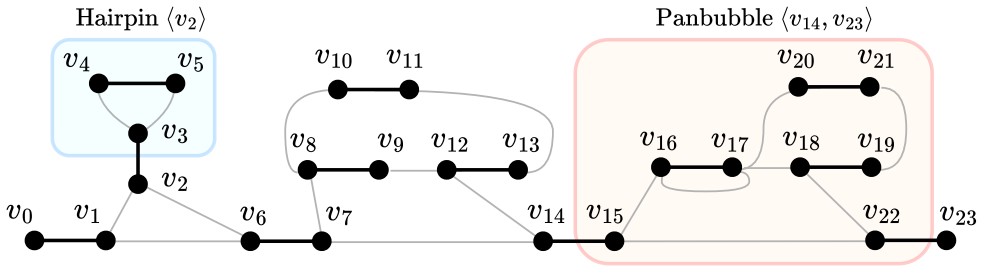

Figure 1: Hairpin $\langle v_2 \rangle$ is highlighted in a blue box. Panbubble $\langle v_{14}, v_{23} \rangle$ is highlighted in a red box. The blue box encloses all vertices in the set $Y(v_2) \setminus \{v_2\}$. The red box encloses all vertices in the set $B(v_{14}, v_{23})$. Note that vertex pairs (v_0, v_7) and (v_6, v_{15}) do not qualify as panbubble entrances because vertex v_2 violates the no-hairpin condition and vertex v_8 violates the contiguity condition, respectively.

Definition (Hairpin). Given a compact biedged graph $G_b = (V_b, E_b, f_b)$, let the set of vertices reachable from vertex $s \in V_b$ without passing through s be denoted by $Y(s)$. A hairpin is said to exist at vertex s if (i) every vertex $v \in Y(s)$ appears in an inverted closed walk starting from s, (ii) removal of all the edges connecting s and $\hat{s}$ separates the subgraph induced by $Y(s) \setminus \{s\}$ from the rest of the graph, and (iii) no vertex in $Y(s)$ other than s satisfies these conditions. The separable subgraph induced by $Y(s) \setminus \{s\}$ is called a *hairpin*.

For any vertex s that satisfies the above conditions, we refer to s and e_s as the *entrance vertex* and the *entrance edge* of the hairpin, respectively. The hairpin is denoted by $\langle s \rangle$.

Definition (Panbubble). A *panbubble* exists between two vertices s and t ($s \neq t$ and $s \neq \hat{t}$) in a compact biedged graph $G_b = (V_b, E_b, f_b)$ if all the following conditions hold:

- *matching*: Let $U(s,t)$ denote the set of vertices reachable from s without passing through s or t. Analogously, define $U(t,s)$ as the set of vertices reachable from t without passing through t or s. The matching condition requires that $U(s,t) = U(t,s)$.
- *separable*: Define $B(s,t)$ as $U(s,t) \setminus \{s,t\}$. If black edges e_s and e_t are removed, the subgraph induced by $B(s,t)$ is disconnected from the rest of the graph.
- *contiguity*: Every vertex $v \in U(s,t)$ appears on at least one walk from s to t.
- *no-hairpin*: No vertex in $B(s,t) \setminus \{\hat{s}, \hat{t}\}$ is an entrance vertex of a hairpin.
- *minimality*: No vertex in $U(s,t)$ other than t forms a pair with s and satisfies the above criteria. Similarly, no vertex in $U(s,t)$ other than s forms a pair with t and satisfies the above criteria.

The separable subgraph induced by $B(s,t)$ is called a *panbubble*. For any set of two vertices $\{s,t\}$ that satisfies these criteria, we denote the panbubble as $\langle s,t \rangle$ (or equivalently $\langle t,s \rangle$). In a panbubble $\langle s,t \rangle$, s and t are called *entrance vertices*, and the edges e_s and e_t are called *entrance edges*. Note that the entrance vertices s,t and the entrance edges e_s, e_t do not belong to the panbubble itself, i.e., they are not part of the subgraph induced by $B(s,t)$. See Figure 1 for an example. In this paper, we seek to solve the following problem:

Problem statement. Given a compact biedged graph $G_b = (V_b, E_b, f_b)$, (i) list the entrance vertex pairs of every panbubble in G_b, and (ii) list the entrance vertices of every hairpin in G_b.

3 Properties of panbubbles and hairpins

In this section, we outline key properties of panbubbles and hairpins, including their possible nesting relationships. These properties will also guide the design of our detection algorithms. For brevity, proofs of all claims in this paper are provided in the Supplementary Document.

bioRxiv preprint doi: https://doi.org/10.1101/2025.11.21.689636; this version posted November 22, 2025. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

Lemma 1. *In a compact biedged graph $G_b = (V_b, E_b, f_b)$, the number of distinct panbubbles can be at most $|V_b/2|$. The number of distinct hairpins can be at most $|V_b|$.*

A valid walk in a biedged graph cannot traverse two gray edges consecutively, which limits the applicability of standard graph-theoretic techniques. To address this limitation, we construct an auxiliary undirected multigraph derived from G_b, in which edge colors are ignored and the walk constraints are relaxed.

Construction of an auxiliary undirected multigraph. We define our auxiliary undirected multigraph $G_U = (V_U, E_U)$ as follows. We assume that G_b is connected (if not, each connected component can be handled independently). First, we copy all the edges and the vertices of G_b into G_U. Note that tips in G_b correspond to vertices that have degree one in G_U. We assume that G_b contains at least one tip*, and hence, G_U has at least one degree-one vertex. Next, we introduce one more vertex v_{src} in G_U, and we connect v_{src} to all degree-one vertices (Figures 2A, 2B). For every vertex v of G_b, we denote the corresponding vertex in G_U by $\bar{v}$. Vertex $\hat{v}$ in G_b corresponds to $\bar{\hat{v}}$ in G_U. The edge of G_U corresponding to a black edge e_v in the biedged graph is denoted by $\bar{e}_v$. Next, for every vertex v of G_b, we ensure that the first vertex in the adjacency list of vertex $\bar{v}$ in G_U is $\bar{\hat{v}}$. We do this to guarantee that every edge in G_U derived from a black edge in G_b appears as a tree edge, rather than a back edge, in any depth-first traversal of G_U.

Unlike biedged graphs, a walk in G_U is defined in the usual sense, that is, a sequence of vertices $(v_0, e_0, v_1, e_1, v_2, \ldots, e_{n-1}, v_n)$ with $n \geq 1$ is a walk in G_U if edge e_i connects the vertices v_i and v_{i+1} for all $0 \leq i \leq n-1$. Walk $(v_0, e_0, v_1, e_1, v_2, \ldots, e_{n-1}, v_n)$ passes through vertices $v_1, v_2, \ldots, v_{n-1}$. A vertex is reachable from $v \in V_U$ if it appears in some walk starting from v. Path and cycle in G_U are also defined in the usual sense. A path is a walk whose vertices are distinct. A cycle in G_U is a closed path that starts and ends at the same vertex. We do not consider a self-loop as a cycle.

Let $T_{v_{src}}$ be the spanning tree obtained by a depth-first traversal of G_U starting from v_{src}. Through this traversal, the edge set E_U is partitioned into a set of tree edges and a set of back edges (Figure 2C). Tree $T_{v_{src}}$ introduces a partial order $\prec$ over the vertices in G_U. For $v_1, v_2 \in V_U$, $v_1 \prec v_2$ if and only if (i) $v_1 \neq v_2$, (ii) v_1 and v_2 belong to a root-leaf path in $T_{v_{src}}$, and (iii) v_1 appears before v_2 in that path from root to leaf. Similarly, for $e_1, e_2 \in E_U$, $e_1 \prec e_2$ if and only if (i) $e_1 \neq e_2$, (ii) e_1 and e_2 belong to a root-leaf path in $T_{v_{src}}$, and (iii) e_1 is traversed before e_2 in that path from root to leaf. A vertex w in V_U is called an ancestor of a tree edge $e = \{u, v\}$ with $u \prec v$ if either $w = u$ or $w \prec u$. Similarly, w is a descendant of edge e if either $w = v$ or $v \prec w$. Next, we define the notion of bracket set and cycle equivalence.

Definition (Bracket Set). A *bracket* of a tree edge e in G_U is a back edge connecting a descendant of e to an ancestor of e. The *bracket set* of e is the set of all brackets of e.

Definition (Cycle-equivalent, Cycle-equivalence classes). Edges e_1 and e_2 are *cycle-equivalent* in G_U if every cycle in G_U contains either both edges or neither edge. The *cycle-equivalence class* of an edge e in G_U is the set of all edges that are cycle-equivalent to e.

We now describe key properties of panbubbles. For every panbubble $\langle s, t\rangle$ in G_b, the vertices in G_U derived from $s, \hat{s}, t, \hat{t}$ lie along a single root-leaf path of tree $T_{v_{src}}$ in a specific order (Lemma 2). The separable condition of $\langle s, t\rangle$ further implies that edges $\bar{e}_s$ and $\bar{e}_t$ must be cycle-equivalent (Lemma 3). If we extract the subgraph of G_b corresponding to panbubble $\langle s, t\rangle$ along with its entrance vertices and edges, then every internal black edge connects one endpoint that has a walk to s and the other that has a walk to t (Lemma 4). Furthermore, the no-hairpin condition of $\langle s, t\rangle$ ensures that no vertex v within a panbubble has a corresponding black edge $\bar{e}_v$ in G_U with an empty bracket set, where $\bar{e}_v$ does not share a common root-leaf path with $\bar{e}_s$ and $\bar{e}_t$ (Lemma 5).

Lemma 2. *For every panbubble $\langle s, t\rangle$ in G_b, either $\bar{s} \prec \bar{\hat{s}} \prec \bar{\hat{t}} \prec \bar{t}$ or $\bar{t} \prec \bar{\hat{t}} \prec \bar{\hat{s}} \prec \bar{s}$ in G_U.*

Lemma 3. *For every panbubble $\langle s, t\rangle$ in G_b, edges $\bar{e}_s$ and $\bar{e}_t$ are cycle equivalent in G_U.*

Lemma 4. *Suppose $\langle s, t\rangle$ is a panbubble in G_b. Consider walks in G_b that do not pass through s or t. For every vertex $u \in U(s, t)$, there exist such walks from s to $\hat{u}$ and from u to t, or there exist such walks from s to u and from $\hat{u}$ to t.*

*In practice, pangenome graphs without tips are rare. Relaxing this assumption is left for future work.

bioRxiv preprint doi: https://doi.org/10.1101/2025.11.21.689636; this version posted November 22, 2025. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

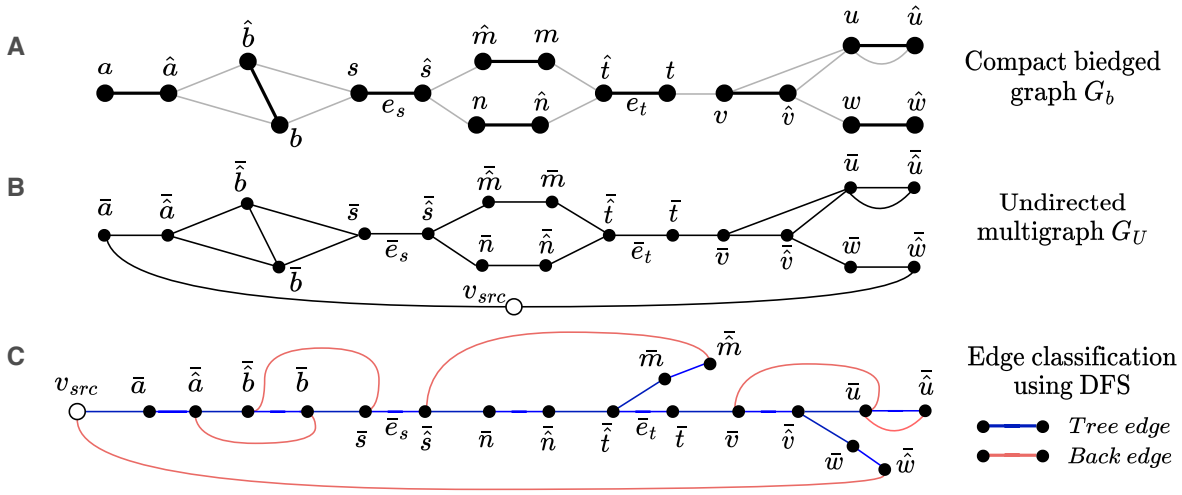

Figure 2: (A) Compact biedged graph G_b. (B) The corresponding undirected multigraph G_U. (C) Partitioning of edges in G_U into a set of tree edges and a set of back edges using DFS starting from vertex v_{src}.

Lemma 5. *In a panbubble $\langle s,t\rangle$ in G_b, there does not exist any vertex $v \in B(s,t) \setminus \{\hat{s},\hat{t}\}$ such that, in G_U, (a) edge $\bar{e}_v$ has an empty bracket set or contains only a single back edge $\{\bar{v},\bar{\hat{v}}\}$ and (b) neither $\bar{e}_s \prec \bar{e}_v \prec \bar{e}_t$ nor $\bar{e}_t \prec \bar{e}_v \prec \bar{e}_s$.*

Next, Theorem 1 states that the above conditions are both necessary and sufficient for a subgraph of G_b to represent a valid panbubble. We omit the minimality constraint here because it will be enforced in our algorithm separately by processing candidate vertex pairs in a suitably chosen order.

Theorem 1. *A vertex pair $s,t \in V_b$ satisfies all the panbubble conditions (excluding minimality) if and only if all the following conditions hold:*

1. *Either $\bar{s} \prec \bar{\hat{s}} \prec \bar{\hat{t}} \prec \bar{t}$ or $\bar{t} \prec \bar{\hat{t}} \prec \bar{\hat{s}} \prec \bar{s}$ in G_U,*
2. *Edges $\bar{e}_s$ and $\bar{e}_t$ are cycle equivalent in G_U,*
3. *Consider walks in G_b that do not pass through s or t. For all vertices $u \in U(s,t) \cup U(t,s)$, there exist such walks from s to $\hat{u}$ and from u to t, or there exist such walks from s to u and from $\hat{u}$ to t.*
4. *There is no vertex $v \in B(s,t)\setminus\{\hat{s},\hat{t}\}$ such that, in G_U, (a) edge $\bar{e}_v$ has an empty bracket set or contains only a single back edge $\{\bar{v},\bar{\hat{v}}\}$ and (b) neither $\bar{e}_s \prec \bar{e}_v \prec \bar{e}_t$ nor $\bar{e}_t \prec \bar{e}_v \prec \bar{e}_s$.*

Similarly, we state necessary and sufficient conditions for a subgraph of G_b to represent a valid hairpin.

Theorem 2. *A vertex s is an entrance vertex of a hairpin in G_b if and only if all the following hold:*

1. *The bracket set of edge $\bar{e}_s$ in G_U is either empty or contains a single back edge connecting $\bar{s}$ and $\bar{\hat{s}}$.*
2. *There does not exist any vertex $v \in V_b \setminus \{\hat{s}\}$ which satisfies the above condition while also satisfying $\bar{s} \prec \bar{v}$ in G_U.*
3. *Consider walks in G_b that do not pass through s. For all vertices $u \in Y(s)$, there exist such walks from s to u and from s to $\hat{u}$.*

Nested relationships between panbubbles and hairpins. Pangenome graphs inherently support nested variation, as illustrated in Figure 3. We characterize the conditions under which the subgraphs of G_b representing (i) two distinct hairpins, (ii) two distinct panbubbles, and (iii) a hairpin and a panbubble, may share a vertex. We first prove that if two panbubbles share a vertex, one must be fully contained within the other (Theorem 3). A panbubble and a hairpin can share a vertex only if the panbubble is completely contained in the hairpin (Theorem 4). Finally, two distinct hairpins can never share a vertex (Theorem 5).

bioRxiv preprint doi: https://doi.org/10.1101/2025.11.21.689636; this version posted November 22, 2025. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

Theorem 3. *Let $\langle x, y\rangle$ and $\langle m, n\rangle$ be two different panbubbles such that $B(x, y) \cap B(m, n) \neq \phi$, then either $B(x, y)$ is a strict subset of $B(m, n)$ or $B(m, n)$ is a strict subset of $B(x, y)$.*

Theorem 4. *Let $\langle x, y\rangle$ and $\langle z\rangle$ be a panbubble and a hairpin respectively in G_b such that $B(x, y) \cap (Y(z) \setminus \{z\}) \neq \phi$, then $B(x, y)$ is a strict subset of $Y(z) \setminus \{z\}$.*

Theorem 5. *Let $\langle m\rangle$ and $\langle n\rangle$ be two distinct hairpins in G_b. Then, the two hairpins do not share any vertex, i.e., $(Y(m) \setminus \{m\}) \cap (Y(n) \setminus \{n\})$ is an empty set.*

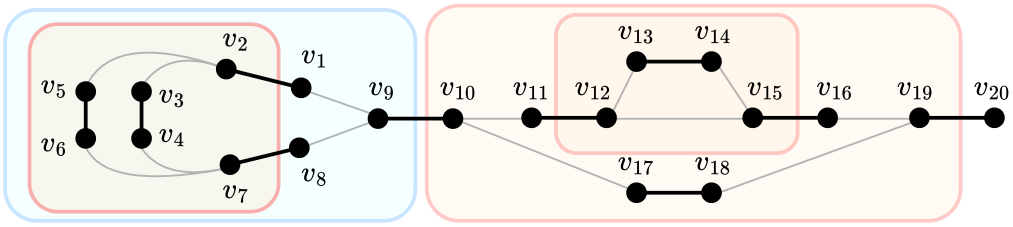

Figure 3: An example illustrating various forms of nested relationships between panbubbles and hairpins. Three red boxes and one blue box are used to highlight three panbubbles and one hairpin, respectively. Panbubble $\langle v_{11}, v_{16}\rangle$ is nested within panbubble $\langle v_9, v_{20}\rangle$. Panbubble $\langle v_1, v_8\rangle$ is nested within hairpin $\langle v_{10}\rangle$.

4 Proposed algorithms to detect panbubbles and hairpins

4.1 Verifying a candidate panbubble entrance vertex pair

Notations and assumptions. Suppose we have a candidate vertex pair (s, t) in G_b for evaluation. We assume that preprocessing has already confirmed that (s, t) satisfies the first and the second condition in Theorem 1. Details of this preprocessing will be discussed later in Section 4.3. We also assume that the following arrays have been precomputed:

- Boolean array A_{bridge} of size $|V_b|$ such that $A_{bridge}[v] = 1$ if and only if the bracket set of edge $\bar{e}_v$ is either empty or it contains only a single back edge $\{\bar{v}, \hat{\bar{v}}\}$.
- Recall that we obtained spanning tree $T_{v_{src}}$ via a depth-first traversal of G_U. Assume that arrays A_d and A_f record the discovery times and the finishing times of all vertices in G_U, respectively. These arrays allow efficient checks of the ancestor-descendant relationships among vertices and edges in $T_{v_{src}}$.

Algorithm. We need to check for the third and fourth condition in Theorem 1. While evaluating a vertex pair (s, t) in G_b, we use two boolean arrays W_s and W_t, each of size $|V_b|$. Array W_s would be used to mark those vertices that have walks originating from them to vertex s without passing through s or t. Similarly, array W_t would be used to mark those vertices that have walks originating from them to vertex t without passing through s or t.

We initialize arrays W_s and W_t with zeroes. First, we do a depth-first traversal starting from vertex s in G_b. During this traversal, we visit all vertices that are reachable from s without passing through s or t. We customize the standard depth-first traversal slightly to account for the walk restrictions in G_b. Consider a single iteration of the depth-first traversal. Suppose the current vertex is $u \in V_b$. We perform the following operations: (i) Mark u as visited, (ii) Update $W_s[\hat{u}]$ to 1, and (iii) If $u \neq \hat{s}$ and $u \neq \hat{t}$, we recursively visit every unvisited neighbor of $\hat{u}$. After completion of the traversal, we repeat the same traversal starting from vertex t to compute array W_t. Subsequently, we declare vertex pair s, t as satisfying all the conditions of a panbubble (excluding minimality) if both the following conditions are satisfied: (a) For all $u \in V_b$, we require $W_s[u] = W_t[\hat{u}] = 1$ or $W_s[\hat{u}] = W_t[u] = 1$ or $W_s[u] = W_s[\hat{u}] = W_t[u] = W_t[\hat{u}] = 0$, and (b) There does not exist any vertex $v \in V_b$ such that (i) $W_s[v]$ or $W_s[\hat{v}] = 1$, (ii) $A_{bridge}[v] = 1$, and (iii) tree edges $\bar{e}_s$, $\bar{e}_v$, $\bar{e}_t$ do not share a root-leaf path in $T_{v_{src}}$.

bioRxiv preprint doi: https://doi.org/10.1101/2025.11.21.689636; this version posted November 22, 2025. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

4.2 Verifying a candidate hairpin entrance vertex

Suppose we have a candidate vertex s in G_b for evaluation. We will reuse the precomputed arrays A_{bridge}, A_d, and A_f defined above. We need to check for the three conditions in Theorem 2. While evaluating s, we allocate a boolean array W of size $|V_b|$, initialized with zeroes. Array W will be used to mark those vertices that have walks originating from them to s without passing through s. To compute this array, we follow the *depth-first* traversal procedure customized for G_b starting from vertex s. During this traversal, we visit all vertices that are reachable from s without passing through s. Consider a single iteration of the traversal. Suppose the current vertex is $u \in V_b$. We perform the following operations: (i) Mark u as visited, (ii) Update $W[\hat{u}]$ to 1, and (iii) If $u \neq \hat{s}$, we recursively visit every unvisited neighbor of $\hat{u}$. After finishing the traversal, we declare vertex s as the entrance vertex of a hairpin if all the following conditions are satisfied: (a) $A_{bridge}[s] = 1$, (b) There does not exist any vertex $v \in V_b \setminus \{\hat{s}\}$ such that $A_{bridge}[v] = 1$ and $\bar{s} \prec \bar{v}$ in G_U, and (c) For all $u \in V_b$, we require either $W[u] = W[\hat{u}] = 1$ or $W[u] = W[\hat{u}] = 0$.

4.3 Complete algorithms

We first construct the undirected multigraph G_U from the compact biedged graph G_b (Section 3). We perform a depth-first traversal of G_U starting from v_{src} to obtain the spanning tree $T_{v_{src}}$ along with arrays A_d and A_f. To compute array A_{bridge}, we remove self-loops and parallel edges in G_U and use Tarjan's linear-time bridge-finding algorithm [29].

Detection of panbubbles (exact algorithm). We partition edges of G_U into cycle-equivalent classes by using the linear-time algorithm of Johnson *et al.* [13]. Next, suppose $C_1, C_2, \ldots, C_k$ are all the cycle-equivalent classes. We process these classes one by one. Suppose the class being processed currently is C_p, $1 \leq p \leq k$. Suppose $e_1, e_2, \ldots, e_n$ are all the distinct black edges in G_b whose corresponding edges in G_U belong to class C_p. We order these edges such that $\bar{e}_i \prec \bar{e}_j$ implies $i < j$. The ordering of edges can be enforced through a level order traversal of $T_{v_{src}}$. Using a pair of nested `for` loops, we test the following for all $1 \leq i < j \leq n$: (i) whether $e_i \prec e_j$ holds, and (ii) whether the vertex pair (v_i, v_j) satisfies the necessary checks discussed in Section 4.1, assuming $e_i = \{v_i, \hat{v}_i\}$ with $v_i \prec \hat{v}_i$ and $e_j = \{v_j, \hat{v}_j\}$ with $\hat{v}_j \prec v_j$. The outer loop iterates over i from 1 to $n-1$, and the inner loop iterates over j from $i+1$ to n. To enforce the minimality condition, we terminate the inner loop early if a panbubble is detected.

Detection of panbubbles (heuristic algorithm). We follow that same procedure as the exact algorithm, except that in the inner loop we restrict consideration to the first edge e_k such that $e_i \prec e_k$, i.e., the edge with the smallest index k satisfying this condition. The remaining edges are not considered inside the inner loop. This design choice is based on our empirical finding that panbubbles rarely include internal edges that are cycle-equivalent to their entrance edges in G_U. However, adversarial cases exist where this assumption fails (Supplementary Figure S4).

Detection of hairpins (exact algorithm). We iterate over all vertices $v \in V_b$ and check whether vertex v satisfies the necessary checks discussed in Section 4.2. This procedure can be paired with either the exact or heuristic panbubble-detection algorithm.

The following theorem states the runtimes of our algorithms.

Theorem 6. *Given a compact biedged graph $G_b = (V_b, E_b, f_b)$, the entrance vertices of all panbubbles and hairpins can be enumerated exactly in $\mathcal{O}(|V_b|^2(|V_b|+|E_b|))$ time. In contrast, the heuristic algorithm performs the same task in $\mathcal{O}(|V_b|(|V_b| + |E_b|))$ time.*

5 Experiments

Experimental setup. We refer to the implementations of our exact and heuristic algorithms as `Billi-exact` and `Billi-heuristic`, respectively. Note that both exact and heuristic implementations employ different algorithms for computing panbubbles; however, the computation of hairpins is performed in the same way. We compare the performance of Billi with Pangene [20] and VG [24], which compute bibubbles and snarls, respectively. We skip comparing with Povu [22] because it did not support graphs with non-numeric vertex

bioRxiv preprint doi: https://doi.org/10.1101/2025.11.21.689636; this version posted November 22, 2025. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

Graph	Before compaction		After compaction		# Components	URL
	# Vertices	# Edges	# Vertices	# Edges		
$G1$	36	66	36	42	1	[17]
$G2$	2,136	4,060	2,136	2,564	1	[18]
$G3$	26,018	51,875	12,344	18,769	22	[15]
$G4$	38,082	63,925	4,162	7,664	1	[16]
$G5$	3,351,876	3,984,038	3,182,154	3,814,316	1	[5]
$G6$	10,434,952	12,437,642	10,268,330	12,271,020	1	[6]
$G7$	185,759,160	221,045,345	183,119,402	218,405,587	25	[4]
$G8$	296,566,820	354,315,094	291,459,944	349,208,218	25	[7]

Table 1: Sizes of the biedged pangenome graphs used in our benchmark.

IDs. All our experiments were done on a dual-socket Intel Xeon Gold 6248R server with 2×24 cores and 750 GB RAM. Among the three tools, only VG supports multi-threading. Accordingly, Billi and Pangene were run using a single thread, and VG was run using 48 threads. Our commands and the tool versions are available in Supplementary Table S1.

Datasets. We used eight publicly available pangenome graphs of varying sizes. For each graph, we report the number of vertices, edges, and connected components under the biedged graph representation in Table 1. Since Billi requires compact biedged graphs as input, it performs graph compaction as a preprocessing step. We also list the sizes of the resulting compacted graphs in Table 1. The first two graphs $G1$ and $G2$ are pangenome graphs constructed by minigraph [19] using 90 C4A/C4B haplotypes and 57 MHC haplotypes, respectively. The next two graphs $G3$ and $G4$ are *gene graphs* constructed using pangene [20]. Graph $G3$ was constructed using 50 *E. coli* genomes, whereas graph $G4$ was constructed using 100 human haplotypes and 10 great ape haplotypes [20]. Graphs $G7$ and $G8$ are the first and second releases of the human pangenome graphs by the Human Pangenome Research Consortium (HPRC) [21]. These graphs were constructed using 94 and 466 assembled human haplotypes, respectively. Graphs $G5$ and $G6$ are subgraphs of $G8$ corresponding to chromosomes Y and X, respectively.

Results. We show the count of bubbles reported by all tools in Table 2. We also report the counts of hairpins identified by Billi. Billi-exact and Pangene were unable to process the larger graphs $G6$-$G8$. Nevertheless, on the first five graphs ($G1$-$G5$), we observe good agreement among the tools. Notably, Billi-exact and Billi-heuristic produced identical output. This implies panbubbles tend to occur within the closest pair of cycle equivalent edges under the $\prec$ ordering (Section 4.3). We also observe that the number of snarls reported by VG is either greater than or equal to the count of panbubbles identified by Billi, whereas the bibubble counts produced by Pangene are generally lower. Figure 4 presents Venn diagrams illustrating the concordance among the tools for graphs $G3$, $G4$, and $G5$.

By definition, a snarl is characterized only by separability and minimality [24]. Figure 5A illustrates a case in which a snarl includes vertices that do not lie on any walk from the source vertex (ID: 208092098) to the sink vertex (ID: 208092203). Because panbubbles satisfy stricter criteria, the set of panbubbles detected by Billi is expected to be a subset of snarls. However, graph $G4$ presents an exception. It contains 7 panbubbles that were not reported as snarls by VG (Figure 4). We further checked and found that these subgraphs agree with the formal definition of a snarl. Supplementary Figure S5 provides visualizations of two such instances. On the other hand, Pangene [20] employs a heuristic implementation that checks every pair of cycle-equivalent edges within a default distance of 100; this occasionally causes it to miss some bibubbles. Furthermore, in graph $G4$, Pangene and VG identify three bubbles that Billi does not report. A closer inspection revealed that these subgraphs contain hairpins (see Figure 5B for an example). Billi reports such hairpins separately (Table 2). One key difference between the bibubble and panbubble formulations is that panbubbles, by definition, do not contain hairpins. Allowing hairpins would break the guaranteed nesting structure among overlapping panbubbles (Theorem 3).

bioRxiv preprint doi: https://doi.org/10.1101/2025.11.21.689636; this version posted November 22, 2025. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

Graph	# Bubbles				# Hairpins
	Billi-exact (panbubble)	**Billi-heuristic** (panbubble)	**VG** (snarl)	**Pangene** (bibubble)	**Billi**
$G1$	4	4	4	4	0
$G2$	376	376	376	374	0
$G3$	403	403	422	400	1
$G4$	409	409	412	398	7
$G5$	372,720	372,720	372,720	369,533	0
$G6$	–	1,649,626	1,649,630	–	2
$G7$	–	29,322,731	29,322,761	–	1
$G8$	–	46,260,770	46,260,820	–	6

Table 2: The number of bubbles reported by Billi, VG, and Pangene. '–' symbol indicates that the tool either crashed or did not finish in 24 hours.

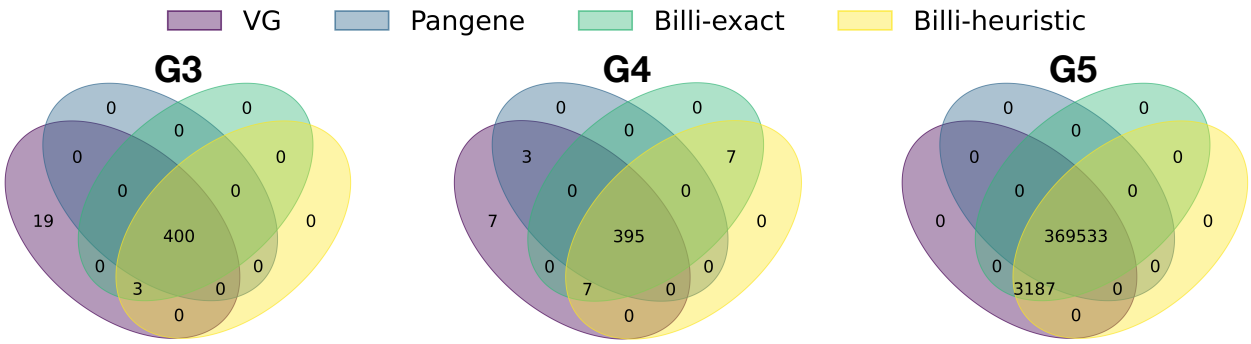

Figure 4: A multi-layer Venn diagram to indicate the count of bubbles detected in pangenome graphs $G3 - G5$. The figure quantifies the degree of overlap among the four bubble detection methods.

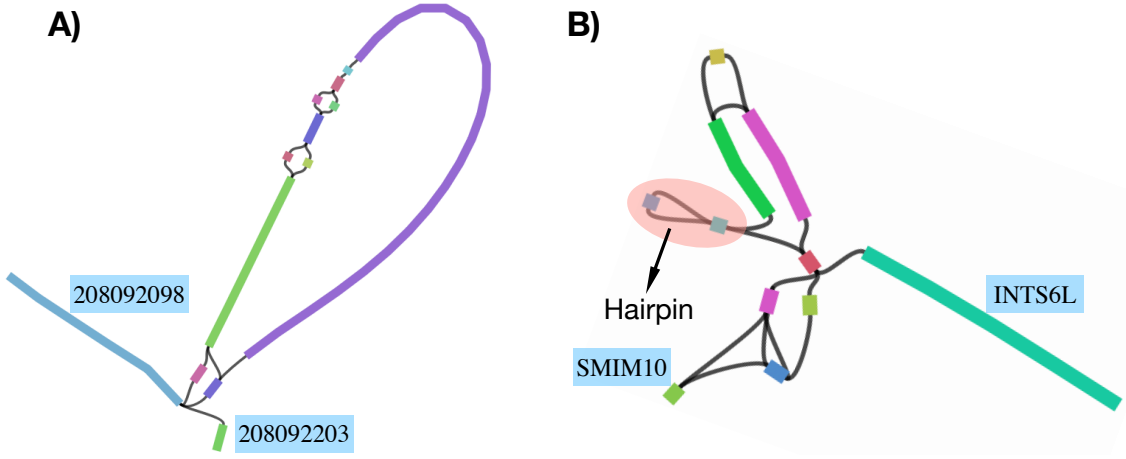

Figure 5: Bandage visualization of two subgraphs. (A) VG marked a snarl in graph $G6$ that is not a panbubble. (B) Pangene identified a bibubble containing a hairpin in graph $G4$.

bioRxiv preprint doi: https://doi.org/10.1101/2025.11.21.689636; this version posted November 22, 2025. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

Table 3 summarizes the runtime and memory usage of all tools. Among the evaluated methods, Billi-heuristic demonstrates the best performance in both runtime and memory footprint. In contrast, Billi-exact fails to complete on graphs $G6$-$G8$. The size of the largest cycle-equivalent class increases dramatically from $G5$ (125,267) to $G6$ (1,390,824). This makes our exact algorithm prohibitively slow. In practice, bubbles are significantly smaller than the full pangenome graph (Supplementary Table S2), which benefits our heuristic algorithm. Billi-heuristic processed the largest graph $G8$ within 75 minutes, whereas other methods either took significantly longer or did not finish.

Graph	Runtime (sec)				Peak memory (MB)			
	Billi-exact	**Billi-heuristic**	**VG**	**Pangene**	**Billi-exact**	**Billi-heuristic**	**VG**	**Pangene**
$G1$	**0.1**	**0.1**	**0.1**	0.3	**3**	**3**	36.9	16.6
$G2$	**0.1**	**0.1**	**0.1**	0.2	4	**3.9**	221.7	27.4
$G3$	0.2	**0.1**	0.3	0.7	10	**9.9**	195.4	78.4
$G4$	**0.2**	**0.2**	0.6	1.3	**10.8**	**10.8**	285.9	144.8
$G5$	12.6	**8.6**	31.3	209.8	1,236	**1,235**	2,173	8,465
$G6$	–	**120**	216	–	–	**4,179**	16,894	–
$G7$	–	**804**	1,710	–	–	**55,404**	160,800	–
$G8$	–	**4,284**	70,164	–	–	**86,579**	490,983.7	–

Table 3: The runtime and peak memory required by various tools. '–' symbol indicates that the tool either crashed or did not finish in 24 hours. Best numbers are highlighted in bold.

6 Conclusions and future work

In this work, we presented new subgraph formulations, called panbubble and hairpin, for bidirected graphs that address the key points that must be respected when feeding these bubble boundaries to a variant caller. Building on these definitions, we then developed Billi, a tool that quickly identifies these structures in large pangenome graphs with hundreds of millions of vertices. As pangenome graphs continue to grow, our results offer a practical and robust algorithmic foundation for discovering novel genomic variation at scale.

Several research directions emerge from this work. First, it remains an open question whether panbubbles and hairpins can be detected using a faster, linear-time algorithm. Second, our current algorithms assume that the input graph contains a degree-one vertex or tip (Section 3). We leveraged this property in our proofs because such vertices are guaranteed not to lie within a panbubble or hairpin, and these vertices serve as a natural entry point for graph traversal. While the assumption holds for most pangenome graphs, we encountered one graph in the Pangene study [20] that has a component with no tips. This case suggests that relaxing the no-tip assumption will be important in the future. A third direction is to investigate how the number and structure of panbubbles evolve as additional haplotypes are augmented into a pangenome graph. Lastly, integrating panbubble and hairpin detection with pangenome-based variant calling or genotyping methods may improve accuracy and offer new biological insights.

Beyond pangenomics, Billi may also be relevant for *de novo* assembly workflows because modern assemblers represent unitig and contig overlaps using bidirected graphs. Detecting panbubbles and hairpins can be useful to isolate subgraphs that require further attention.

Acknowledgments

We thank Paul Medvedev and Heng Li for providing useful feedback. We also thank the Human Pangenome Reference Consortium (HPRC) for openly sharing their data. This research is supported in part by funding from the DBT/Wellcome Trust India Alliance (IA/I/23/2/506979). We used computing resources provided by the National Energy Research Scientific Computing Center (NERSC), USA.

bioRxiv preprint doi: https://doi.org/10.1101/2025.11.21.689636; this version posted November 22, 2025. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

References

[1] Jasmijn A Baaijens, Paola Bonizzoni, Christina Boucher, Gianluca Della Vedova, Yuri Pirola, et al. "Computational graph pangenomics: a tutorial on data structures and their applications". In: *Natural computing* 21.1 (2022), pp. 81–108.

[2] Ljiljana Brankovic, Costas S Iliopoulos, Ritu Kundu, Manal Mohamed, Solon P Pissis, et al. "Linear-time superbubble identification algorithm for genome assembly". In: *Theoretical Computer Science* 609 (2016), pp. 374–383.

[3] Sai Chen, Peter Krusche, Egor Dolzhenko, Rachel M Sherman, Roman Petrovski, et al. "Paragraph: a graph-based structural variant genotyper for short-read sequence data". In: *Genome biology* 20.1 (2019), p. 291.

[4] HPRC Consortium. *HPRC_v1*. `https://s3-us-west-2.amazonaws.com/human-pangenomics/pangenomes/freeze/freeze1/minigraph-cactus/hprc-v1.1-mc-chm13/hprc-v1.1-mc-chm13.gfa.gz`. Online; accessed 1 Oct 2025. 2025.

[5] HPRC Consortium. *HPRC_v2*. `https://human-pangenomics.s3.amazonaws.com/pangenomes/scratch/2025_02_28_minigraph_cactus/hprc-v2.0-mc-chm13/hprc-v2.0-mc-chm13.chroms/chrY.vg`. Online; accessed 1 Oct 2025. 2025.

[6] HPRC Consortium. *HPRC_v2*. `https://human-pangenomics.s3.amazonaws.com/pangenomes/scratch/2025_02_28_minigraph_cactus/hprc-v2.0-mc-chm13/hprc-v2.0-mc-chm13.chroms/chrX.vg`. Online; accessed 1 Oct 2025. 2025.

[7] HPRC Consortium. *HPRC_v2*. `https://human-pangenomics.s3.amazonaws.com/pangenomes/scratch/2025_02_28_minigraph_cactus/hprc-v2.0-mc-chm13/hprc-v2.0-mc-chm13.gfa.gz`. Online; accessed 1 Oct 2025. 2025.

[8] Fawaz Dabbaghie, Jana Ebler, and Tobias Marschall. "BubbleGun: enumerating bubbles and superbubbles in genome graphs". In: *Bioinformatics* 38.17 (2022), pp. 4217–4219.

[9] Jack Edmonds and Ellis L Johnson. "Matching: A well-solved class of integer linear programs". In: *Combinatorial Optimization—Eureka, You Shrink! Papers Dedicated to Jack Edmonds 5th International Workshop Aussois, France, March 5–9, 2001 Revised Papers*. Springer. 2003, pp. 27–30.

[10] Erik Garrison, Andrea Guarracino, Simon Heumos, Flavia Villani, Zhigui Bao, et al. "Building pangenome graphs". In: *Nature Methods* (2024), pp. 1–5.

[11] Fabian Gärtner, Lydia Müller, and Peter F Stadler. "Superbubbles revisited". In: *Algorithms for Molecular Biology* 13.1 (2018), p. 16.

[12] Glenn Hickey, Jean Monlong, Jana Ebler, Adam M Novak, Jordan M Eizenga, et al. "Pangenome graph construction from genome alignments with Minigraph-Cactus". In: *Nature biotechnology* 42.4 (2024), pp. 663–673.

[13] Richard Johnson, David Pearson, and Keshav Pingali. "The program structure tree: Computing control regions in linear time". In: *Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation*. 1994, pp. 171–185.

[14] Brice Letcher, Martin Hunt, and Zamin Iqbal. "Gramtools enables multiscale variation analysis with genome graphs". In: *Genome biology* 22.1 (2021), p. 259.

[15] Heng Li. *Pangene graphs*. `https://zenodo.org/records/14854301/files/Ecoli-v1.1-p0g30r5.gfa.gz`. Online; accessed 1 Oct 2025. 2025.

[16] Heng Li. *Pangene graphs*. `https://zenodo.org/records/14854301/files/human100p10-v1.1-a1.gfa.gz`. Online; accessed 1 Oct 2025. 2025.

[17] Heng Li. *Sample graphs and sequences for testing sequence-to-graph alignment*. `https://zenodo.org/records/6617246/files/C4-90.gfa.gz`. Online; accessed 1 Oct 2025. 2022.

bioRxiv preprint doi: https://doi.org/10.1101/2025.11.21.689636; this version posted November 22, 2025. The copyright holder for this preprint (which was not certified by peer review) is the author/funder, who has granted bioRxiv a license to display the preprint in perpetuity. It is made available under aCC-BY 4.0 International license.

[18] Heng Li. *Sample graphs and sequences for testing sequence-to-graph alignment*. `https://zenodo.org/records/6617246/files/MHC-57.gfa.gz`. Online; accessed 1 Oct 2025. 2022.

[19] Heng Li, Xiaowen Feng, and Chong Chu. "The design and construction of reference pangenome graphs with minigraph". In: *Genome biology* 21 (2020), pp. 1–19.

[20] Heng Li, Maximillian Marin, and Maha R Farhat. "Exploring gene content with pangene graphs". In: *Bioinformatics* 40.7 (2024), btae456.

[21] Wen-Wei Liao, Mobin Asri, Jana Ebler, Daniel Doerr, Marina Haukness, et al. "A draft human pangenome reference". In: *Nature* 617.7960 (2023), pp. 312–324.

[22] Njagi Mwaniki, Erik Garrison, and Nadia Pisanti. "Popping bubbles in pangenome graphs". In: *arXiv preprint arXiv:2410.20932* (2024).

[23] Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. "Detecting superbubbles in assembly graphs". In: *International workshop on algorithms in bioinformatics*. Springer. 2013, pp. 338–348.

[24] Benedict Paten, Jordan M Eizenga, Yohei M Rosen, Adam M Novak, Erik Garrison, et al. "Superbubbles, ultrabubbles, and cacti". In: *Journal of Computational Biology* 25.7 (2018), pp. 649–663.

[25] Amatur Rahman and Paul Medvedev. "Assembler artifacts include misassembly because of unsafe unitigs and underassembly because of bidirected graphs". In: *Genome Research* 32.9 (2022), pp. 1746–1753.

[26] Jonas Andreas Sibbesen, Lasse Maretty, Danish Pan-Genome Consortium, and Anders Krogh. "Accurate genotyping across variant classes and lengths using variant graphs". In: *Nature genetics* 50.7 (2018), pp. 1054–1059.

[27] Jouni Sirén, Jean Monlong, Xian Chang, Adam M Novak, Jordan M Eizenga, et al. "Pangenomics enables genotyping of known structural variants in 5202 diverse genomes". In: *Science* 374.6574 (2021), abg8871.

[28] Wing-Kin Sung, Kunihiko Sadakane, Tetsuo Shibuya, Abha Belorkar, and Iana Pyrogova. "An $O(m \log m)$-time algorithm for detecting superbubbles". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 12.4 (2014), pp. 770–777.

[29] R. Endre Tarjan. *A Note on Finding the Bridges of a Graph*. Tech. rep. UCB/ERL M427. Feb. 1974. URL: `http://www2.eecs.berkeley.edu/Pubs/TechRpts/1974/29303.html`.

[30] Ryan R Wick, Mark B Schultz, Justin Zobel, and Kathryn E Holt. "Bandage: interactive visualization of de novo genome assemblies". In: *Bioinformatics* 31.20 (2015), pp. 3350–3352.