PanG2G: Scalable Base-Pair-Resolution Graph-to-Graph Alignment of Pangenome Graphs

Ahmad Shakleya and Suhaib A. Fahmy
King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia.
ahmad.shakleya@kaust.edu.sa

***Abstract*—Pangenome variation graphs have replaced the single linear reference as the state of the art for representing human genetic diversity, yet a basic operation is still missing: directly comparing two such graphs to each other. On population pangenomes the delta graph shows the population-specific structural variation that a sequence-to-reference method cannot express without a linear projection. We present PanG2G, the first system for base-pair resolution graph-to-graph alignment of pangenome variation graphs. In this work we characterize the workload, describe the system and validate the reference-anchored output on a planted ground truth. The reference-anchored diff scales near-linearly to an 8.2×10^9 bp graph pair ($\sim2.6\times$ the human genome) on a single core in 72 min with 100% precision/recall, and the parallel snarl-pairing engine reaches $51.9\times$ DP-phase speedup on 64 cores.**

***Index Terms*—comparative genomics, pangenome, variation graph, graph alignment, structural variation**

I. Introduction

For years, researchers have compared every human genome to a single linear reference, and the sequences missing from that reference are not evenly spread worldwide. Instead, they are concentrated in populations that were not included when the reference was created. Deep sequencing of 910 people of African descent identified 296 megabases of sequence that are missing from GRCh38 [1]. The Human Pangenome Reference Consortium (HPRC) produced the first broad solution to this issue: a draft assembled from 47 diploid genomes [2]. Population-specific graphs have since followed. For example, phased assemblies of 10 Japanese and 9 Saudi individuals now provide each population with its own graph, and each of those graphs recovers more variants in that population than the linear reference would [3]. Much of this private variation is structural, and structural variants (SVs) change more nucleotides per genome than single-nucleotide polymorphisms (SNPs) and small insertions and deletions (indels) [4].

The natural next question is comparative: given the graphs of two populations, which alleles are present in only one of them? Answering it means aligning the two graphs allele by allele, and no tool does this. Two workarounds exist, and both give up what pangenomes gained. The first method projects each graph onto a linear reference as a VCF and compares the two VCFs, which reintroduces the reference bias [2], [5] and hides every allele that the reference lacks. The second method merges the two graphs into one larger graph, which requires all of the underlying assemblies and must be redone whenever either graph changes. Aligning a sequence to a sequence [6], [7] or to a graph [8]–[10] is well established. Aligning one population graph to another at base-pair resolution is not, and that is what this paper addresses.

In general, graph-to-graph alignment includes subgraph isomorphism and is therefore an NP-hard problem [11]. The key observation is that variation graphs have a particular structure. Wherever haplotypes diverge, the graph forms a bubble: the paths split at a common node, follow different sequences, and rejoin at another common node (Fig. 1). Each path through the bubble is an allele, and the bubble itself is called a snarl [12]. Bubbles can be nested within bubbles, so the snarls of a graph form a tree, and in practice this tree has bounded tree-width [13].

We make use of this insight to develop PanG2G, a system that efficiently computes these graph-to-graph alignments. The workload is irregular: most of the snarls are very small SNP bubbles, but a few structural-variant regions are extremely large, so the amount of work per snarl has a heavy-tailed distribution and a static work partition would result in an unbalanced load.

PanG2G factors the alignment into independent per-snarl bipartite-matching subproblems, each solved optimally in $O(k^3)$ with the Jonker-Volgenant linear assignment solver (LAPJV) [14], which finds the minimum-cost one-to-one matching between the two snarls' alleles. Snarl pairing, however, is greedy, so end-to-end optimality is not guaranteed.

The contributions made in this paper include:

- We define base-pair-resolution graph-to-graph alignment for variation graphs and introduce PanG2G, which is the first system to address this problem. The delta graph that it produces is the graph-level equivalent of a VCF in that it contains every allele found in one population graph but not in the other.
- Profiling the reference prototype shows that per-snarl work is heavy-tailed (Gini > 0.55) and scale-invariant.
- We have developed an engine using a van Emde Boas snarl-tree layout (vEB), a minimizer anchor index, exact LAPJV bipartite matching, and an OpenMP priority-dispatch runtime.
- We show a reference-anchored diff that scales almost linearly beyond whole-genome size and, on real African and East Asian (1000 Genomes) graphs, a delta significantly enriched near gnomAD-SV and ClinVar pathogenic sites ($p < 10^{-3}$).

- We release PanG2G, its prototype, generators and experiment harnesses as open source[1].

II. Related Work

Aligning two pangenome graphs is a global alignment problem: every allele in one graph is either aligned to an allele in the other or is a gap. For two strings this was solved by Needleman and Wunsch [6], whose $O(mn)$ dynamic programming algorithm is still the basis for global alignment. Partial order alignment (POA) [15] extended that concept from strings to directed acyclic graphs and can even align one partial order graph to another. The problem with POA is that its alignment table has one cell for every pair of nodes, which is tractable for gene-sized alignments but not for pangenomes with 10^7 nodes. Speed is not the only issue. Optimal alignment of two multiple alignments is already NP-complete [16], and so is optimal alignment of a single sequence to an edited graph [17]. Scalable methods have to make use of the graph structure, and PanG2G does this by using snarls. Read mappers sidestep the problem entirely by keeping one side a string (mapping to a sequence [18] or to a graph [19], [20]). We mention them only because the linearize-and-map baseline of Section IV, which expands each graph into its haplotypes and maps every haplotype separately, is built on them.

Comparing arbitrary graphs is an older problem outside genomics, and it offers a useful trick. The exact graph edit distance is NP-hard, so the usual approach is to approximate it by reducing it to a bipartite assignment of small local substructures, which can be solved using Hungarian-type algorithms [21]. However, this reduction does not take into account the bubble structure of variation graphs or report alleles at the base-pair level. PanG2G uses the same assignment approach but restricts it to paired snarls, where the problem is small enough to solve exactly.

Building pangenome graphs is now routine, with vg [19], minigraph [22], Minigraph-Cactus [23] and PGGB [24]. Comparing two of them is not. The usual route projects each graph onto a reference path as a VCF with `vg deconstruct`, as in the HPRC release [2], and compares the VCFs. The alternatives are coarser. ODGI [25] measures path similarity within a single graph, while PanGraph [26] and SplitMEM [27] compare pangenomes at the level of gene blocks or suffix structures rather than alleles. The community has anticipated the missing step: the Pangenome Graph Variation Format (PGVF) [28], a fork of GFA, was proposed to store graph-to-graph alignments. To our knowledge, PanG2G is the first system that computes such alignments at the allele level.

III. Methodology

A. Problem Definition

A variation graph G is a directed acyclic graph (DAG) with nodes labeled by sequences σ, and its haplotypes are paths from source to sink. When haplotypes differ, the graph contains a bubble, called a snarl $S=(s,t)$, which is a minimal pair of boundary nodes that separates the subgraph between them from the rest of G [12]. Snarls are nested to form a snarl tree $T(G)$, and each $s\to t$ path (the concatenation of the internal node labels) represents one of the snarl's k alleles.

Aligning two graphs G_A and G_B means finding which parts of the graphs correspond to each other. An alignment $\mathscr{M}=(M_v,M_e,M_s)$ consists of partial one-to-one mappings between the nodes, edges, and snarls of G_A and G_B, subject to two conditions: (i) M_s preserves the ancestry between $T(G_A)$ and $T(G_B)$ and the order of snarls along the matched backbone M_v; and (ii) each snarl pair $(S,S')\in M_s$ carries a partial one-to-one allele matching $\mu_{S,S'}$. The goal is to find $\mathscr{M}^*=\arg\max_{\mathscr{M}} W(G_A,G_B,\mathscr{M})$ where

$$W(G_A,G_B,\mathscr{M})=\alpha\sum_{(u,v)\in M_v}\mathrm{sim}(\sigma_A(u),\sigma_B(v))+\beta\sum_{(e,e')\in M_e}\mathbf{1}[\text{cons.}]+\gamma\sum_{(S,S')\in M_s}\mathrm{SimBubble}(S,S'). \quad (1)$$

Here, $\mathrm{sim}(s_1,s_2)=1-\mathrm{edit}(s_1,s_2)/\max(|s_1|,|s_2|)$, and $\mathbf{1}[\text{cons.}]$ rewards a matched edge whose endpoints are matched. $\mathrm{SimBubble}(S,S')$ sums the similarity of the pairs in $\mu_{S,S'}$ minus a gap penalty for each unmatched allele. The delta graph $\Delta_{A\to B}$ (or $\Delta_{B\to A}$) contains the alleles of G_A (or G_B) that are not matched by $\mathscr{M}^*$, including every allele of a snarl without a partner. Each entry records its exact inserted or deleted sequence (see Fig. 1), making the delta a graph-level analog of a VCF. No sequence-to-reference pipeline can produce this without first projecting both graphs onto a linear backbone. The matched elements make up the shared alignment, which is written as a GAF-style file instead of being materialized as a graph.

Condition (i) is what makes the problem manageable. Without it, the problem involves subgraph isomorphism and becomes NP-hard [11]. With it, the optimum decomposes over the snarl tree, one snarl pair at a time (the recurrence is given as Eq. (2) in §III-B), and each leaf becomes an assignment problem. PanG2G has two modes. In the default *snarl-pairing mode*, the snarl map M_s is found by a heuristic (§III-B2) and the alleles within each pair are matched exactly, so the matching within each snarl is optimal but the overall result may not be $\mathscr{M}^*$. If both graphs embed a common reference, the *reference-anchored mode* (§III-B4) fixes M_s by reference coordinate instead and matches alleles only when they are identical. Its output is the *reference-anchored diff*.

B. The Algorithm

In snarl-pairing mode, PanG2G finds anchors, breaks each graph into its snarl tree, pairs snarls between the graphs, matches alleles within each pair, and outputs the deltas, the shared alignment, and a score (Fig. 2, Algorithm 1). If there were nested children, the score would decompose recursively,

$$\mathrm{OPT}(S_A,S_B)=W_{\text{seq}}+W_{\text{topo}}+\max_{\mu}\sum_{(c,c')\in\mu}\mathrm{OPT}(c,c'). \quad (2)$$

Here, μ refers to child-snarl matchings. However, the implementation and evaluation are intentionally limited to depth-1. While nested snarls are detected, only top-level snarls are

[1]PanG2G Repository: https://github.com/accl-kaust/pang2g

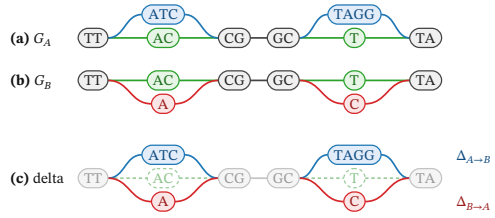


Fig. 1. The delta at base-pair resolution. Each pill is a node storing a string of base pairs, and each bubble a snarl. Graphs (a) G_A and (b) G_B share one allele per snarl and each adds its own. (c) The delta keeps the alleles present in exactly one graph, $\Delta_{A\to B}$ from G_A and $\Delta_{B\to A}$ from G_B, each with its exact sequence. Matched alleles (dashed) go to the shared alignment and the backbone.

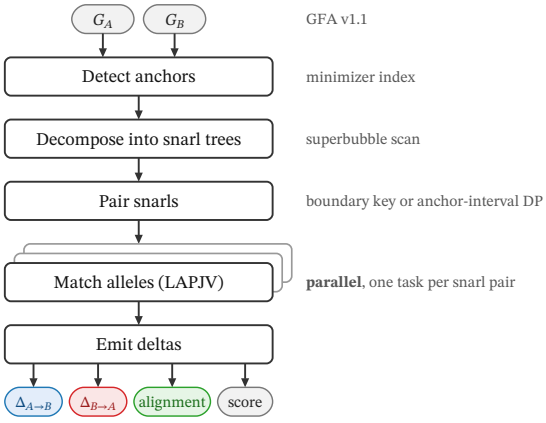


Fig. 2. The PanG2G pipeline. Each stage is annotated with its method. The stacked cards mark the one stage that runs in parallel, one task per snarl pair under the priority-dispatch scheduler of Algorithm 1. The outputs are the two deltas, the shared alignment (GAF) and the score W.

used, and the graphs we test (from `vg construct` and our generators) have only simple SNP and indel bubbles. Each snarl pair is therefore one independent assignment problem. The recurrence Eq. (2), the van Emde Boas layout and the task DAG are implemented for nested trees but used only in this flat form. The weights (α,β,γ) are set to $(1,0.5,2)$ by default, giving more weight to the structural term because allele identity is the main signal for the delta. We do not tune these weights: the delta depends only on which alleles are matched (μ), and scaling γ changes the score but not which allele pairs rank highest.

1) Anchor detection: Anchors are points where both graphs clearly agree: node pairs with in- and out-degree at most one in both graphs and nearly identical sequences (sim ≥ 0.9). Comparing every node to every other would require about 10^{14} comparisons at human scale, so PanG2G indexes eligible nodes in one graph using minimizers (FNV-1a hashes with a minimap2-style window [18]; a build-time option can switch back to the naive scan). Candidates that share a minimizer are checked for full similarity and resolved greedily. Nodes shorter than the 15-bp k-mer are never anchored. The degree rule allows allele nodes of simple bubbles and the backbone, so in our `vg construct` graphs, half of all nodes qualify, and over 80% of these are 1-bp SNP alleles. The anchors

Algorithm 1 PanG2G alignment (**depth-1 form**, as implemented and evaluated). $W_{\text{seq}} = \alpha\cdot$ mean similarity of the two snarls' source and sink nodes; $W_{\text{topo}} = \beta \cdot \mathbf{1}$[both are simple bubbles].

Require: graphs G_A, G_B; weights α,β,γ; gap g; floor τ
Ensure: deltas $\Delta_{A\to B}, \Delta_{B\to A}$, shared alignment H, score W

$\mathcal{N} \leftarrow$ FindAnchors(G_A, G_B) ▷ minimizer index
$T_A \leftarrow$ Decompose(G_A); $T_B \leftarrow$ Decompose(G_B)
$P \leftarrow$ PairSnarls$(T_A, T_B, \mathcal{N})$ ▷ boundary-key or anchor-interval rule
$\mathcal{T} \leftarrow$ BuildTaskDAG(P) ▷ priority $= \sum k_i^3$
parfor $(S_A, S_B) \in \mathcal{T}$ **by priority dispatch do**
  $a \leftarrow$ Alleles(S_A); $b \leftarrow$ Alleles(S_B)
  $C_{ij} \leftarrow 1 - \text{sim}(a_i, b_j)$; clamp $C_{ij} > 1-\tau$ to g
  $(s, \mu) \leftarrow$ LAPJV(C) ▷ $O(k^3)$; greedy if $k>30$
  $w[S_A, S_B] \leftarrow W_{\text{seq}} + W_{\text{topo}} + \gamma s$; record μ
end parfor
$W \leftarrow \sum w[S_A, S_B] + \alpha \sum_{(u,v)\in\mathcal{N}} \text{sim}(u, v)$
$U_A, U_B \leftarrow$ unmatched alleles $\cup$ alleles of unpaired snarls
$\Delta_{A\to B}, \Delta_{B\to A} \leftarrow$ BuildDeltas(U_A, U_B)
$H \leftarrow \mathcal{N} \cup P \cup \{\mu\}$ ▷ shared alignment (GAF)
return $\Delta_{A\to B}, \Delta_{B\to A}$, H, W

that matter are the roughly 15% of eligible nodes that are at least 15 bp long: these are the reference segments between variants, which `vg construct` cuts into nodes of at most 32 bp. Anchors contribute to the α term in Eq. (1), and only the anchor-interval rule below uses them to align the two coordinate systems. Anchoring is one-to-one at the node level, so a reference segment that the two graphs cut into nodes of different lengths is never anchored. The reference-anchored mode does not use anchors, so it is unaffected.

2) Snarl decomposition and pairing: Snarl boundaries can come from `vg snarls` [19], but by default PanG2G runs a built-in superbubble scan after Onodera et al. [29]. Starting from each node with more than one successor, it walks forward in topological order and adds a node only once all of that node's predecessors have been seen. The first node at which every walked path has rejoined closes the snarl. Each scan costs only the size of its bubble, so decomposition is near-linear ($N^{1.07}$ up to 4 Gbp, §IV-C). Both backends then enumerate every $s \to t$ path as an allele.

The heuristic is applied during pairing, and PanG2G uses two pairing rules. The main rule, called the *boundary-key rule*, uses the source and sink sequences of a snarl as its boundary key. Two snarls are paired if they share the same boundary key. Snarls that share a key are consumed in topological order. This rule is linear and exact for unique keys, but keys can repeat: 10-13% of bubbles on `vg construct` graphs share a key, since 35-40% of boundary nodes are at most 2 bp. If a snarl is present in only one graph, it shifts every later snarl with that key by one position. The alternative *anchor-interval rule* (`--pairing interval`) avoids this drift: it chains the anchors in topological order and pairs the snarls between consecutive anchors using an

order-preserving Needleman-Wunsch algorithm over the two snarl sequences, scoring matches by the fraction of identical alleles plus boundary similarity. Because the alignment restarts at every anchor, a pairing error cannot propagate past one, and within each interval the pairing is optimal. The cost is the product of the two interval lengths summed over intervals, $\sum_i |A_i||B_i|$, which stays near-linear when anchors are dense. Q1 (§IV-A) compares the two rules, and under either one an unpaired snarl sends all its alleles to the delta.

3) Per-snarl bipartite matching: For each pair, PanG2G looks for the partial allele matching with the highest similarity. It creates a $k \times k$ cost matrix $C_{ij} = 1 - \mathrm{sim}(a_i, b_j)$, sets pairs below the similarity threshold $\tau = 0.75$ to the gap cost so that a gap is preferred over a poor match, pads rectangular cases with gap-cost dummies, and solves the assignment exactly using Jonker-Volgenant (LAPJV) in $O(k^3)$ [14], optimizing matches and gaps in one step. For snarls with $k > 30$, it uses a greedy $1/2$-approximation, and a brute-force method is used only as an oracle for $k \leq 8$.

4) Reference-anchored mode: When both graphs share a common reference, as is typical for `vg construct` and Minigraph-Cactus graphs, the reference-anchored mode skips anchor detection and snarl pairing, matching instead on reference coordinates. Each bubble is split into atomic SNPs and indels, each identified by its reference position (*bubble_open*+*offset*). The reference-anchored diff reports atomic variants whose key appears in only one graph. Because the key depends only on the coordinate, a merged bubble in one graph produces the same atomic variants as separate bubbles in another. This mode is not affected by node boundaries, but it does depend on how variants are represented. If two graphs have the same indel at different positions, such as left- or right-aligned in a homopolymer, the keys will differ and a false delta will be reported, as shown in the HLA experiment in Section §IV-A. PanG2G does not normalize variants, so graphs from the same tool are consistent, but graphs from different tools are not. Normalization is needed for graphs from different sources. Left-alignment is just one part of normalization, since different builders like Minigraph-Cactus and PGGB have their own rules for cutting, nesting, and merging variation. Hence, all comparisons are between graphs from the same builder. Comparing graphs from different builders would require breaking down each allele into a standard set of primitive variants before diffing, which is left for future work.

C. Workload Profiling

Three findings from profiling the reference prototype, using synthetic graphs with snarl sizes following a Zipf law based on HPRC statistics, influenced the design of the engine.

Finding 1: matching cost explodes with snarl size. Brute-force enumeration ($O(k!)$) works quickly for SNP bubbles but becomes impractical for k greater than about 8, taking 57 ms per pair at $k = 8$ in C++ and 2 s in Python. LAPJV takes 1.2 ms even at $k = 50$ (see Fig. 3b), so exact matching remains practical for structural-variant regions with k between 20 and 50. The default engine nevertheless keeps a conservative threshold: snarls with $k \leq 30$ are matched exactly with LAPJV, and snarls with $k > 30$ use the greedy approximation.

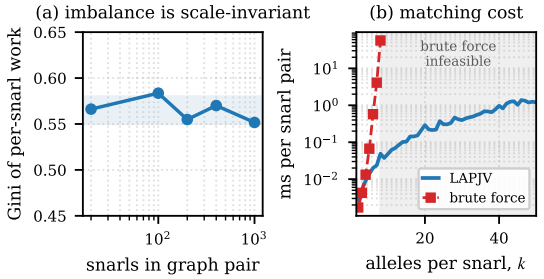


Fig. 3. (a) The Gini coefficient of estimated per-snarl work (k^3) stays above 0.55 across graph sizes (reference prototype, HPRC-calibrated synthetic graphs), so static partitioning is load-imbalanced at every scale. (b) Cost of per-snarl matching versus allele count k.

Finding 2: the workload is heavy-tailed and scale-invariant. About 67% of snarls are $k = 2$ SNP bubbles, but because estimated work grows as k^3, a few large snarls dominate the workload. The Gini coefficient of estimated per-snarl work stays at 0.55-0.58 from 20 to 1000 snarls (Fig. 3a). Real graphs contain 10^6 to 10^7 snarls, far more than we profiled, but since the coefficient is flat over two orders of magnitude we expect the same imbalance at full scale, where a static equal-count partition would leave a few threads with most of the work.

Finding 3: memory access, not arithmetic, is the scaling wall. The bottom-up traversal reads allele sequences from both graphs for each snarl pair. At human scale this working set runs to hundreds of gigabytes (§IV-C), far beyond the last-level cache, so with a naive layout every snarl pair is served from main memory (DRAM), and data locality matters more than arithmetic.

D. System Design and Implementation

PanG2G is implemented in C++20, with a Python reference prototype used as a correctness oracle. The prototype pairs snarls greedily over all pairs, which is quadratic in complexity, while the main engine uses the linear rules described above. It reads GFA v1.1 (S/L/P and W lines). To address Finding 3, all node sequences are stored in a single flat byte array indexed by (offset, length), and adjacency is stored in CSR format, so the DP kernel can stream sequence data without pointer chasing. Unless otherwise noted, experiments are run on a single-socket AMD EPYC 7702 with 64 cores, 4 NUMA domains, 512 GiB DDR4-3200, 256 MiB L3 cache, running Ubuntu 20.04 and GCC 9.4-O3.

Structure-aware layout. Each snarl's DP table and alleles are stored together in one contiguous memory block, which benefits all workloads. For nested trees, where breadth-first storage scatters sibling subtrees and a bottom-up traversal causes cache thrashing beyond L3, PanG2G arranges the tree in van Emde Boas order [30], [31]. It stores a subtree of size S in $O(S/B)$ contiguous lines for $O(N/B)$ cache misses overall. For the depth-1 trees evaluated here, vEB order is the same as BFS order, so this feature is mainly for future

nested Minigraph-Cactus inputs, as confirmed by the ablation (§IV-D).

Priority-dispatch runtime. The parallel loop of Algorithm 1 (lines 5-10) is the *DP phase*. Everything before it, namely parsing, index build and decomposition, is the *serial setup*. Finding 2 shows that a few snarl pairs account for most of the work, so the scheduler must assign those first. PanG2G uses the longest-processing-time-first method (LPT) [32] with OpenMP: it keeps all snarl pairs in a single priority queue, ordered by their estimated work ($\sum k_i^3$). When a thread is free, it takes the largest remaining snarl pair. This queue is shared and protected by a mutex. The usual alternative, which is one work-stealing deque per thread, avoids the lock but lets idle threads take any task, which breaks the LPT order we need. The downside of the single queue is lock contention, which could limit scalability beyond the 64 cores we tested (§IV-B). Running in parallel does not change the output: the delta matches the sequential aligner byte-for-byte, and the score agrees within 10^{-3}, with any difference due to floating-point summation order.

IV. Evaluation

We evaluate whether PanG2G finds the correct alleles (Q1), how fast and scalable it is (Q2), which design choices are most important (Q3), and if the delta captures biologically relevant variation (Q4).

A. Q1: Validation

We add shared, A-only, and B-only alleles to graph pairs, then check whether the deltas recover them. We focus on three properties: (i) determinism, whether the parallel engine matches the sequential engine; (ii) optimality under the scoring scheme; and (iii) biological truth, or whether (α, β, γ) matches the correct mapping. The last point is not formally proven. We score a delta by comparing its allele precision and recall to the planted set, using the F1 score as a summary.

Planted graphs. For pairs with 5 to 10^5 snarls (two shared and one private allele per side, 8-bp boundary nodes, three seeds each), we achieve a precision of 1.000 and recall between 0.997 and 1.000. In each paired snarl, LAPJV recovers the planted alleles exactly, and shared alleles never appear in the delta. Both pairing rules recover the planted deltas almost exactly. Any loss comes from the scoring model, not the pairing itself. A few random 12-mers fall within the $\tau = 0.75$ similarity threshold and are matched.

Pairing error at scale. We create the cascade on planted pairs with 1- or 2-bp boundary nodes so that keys do collide, as they would in real `vg construct` graphs, with 10% of the snarl slots private to one graph and a shared 24-bp linker node (an anchor) after every fourth snarl. With 1-bp boundaries, the delta F1 of the boundary-key rule drops from 1.00 at 5 snarls to 0.82 at 50, 0.63 at 10^3, and 0.55 at 10^5 (precision 0.38 at recall 0.998). With 2-bp boundaries, it drops from 0.99 at 50 to 0.57 at 10^5, while with unique 8-bp keys it stays at $F1 \geq 0.9995$ (one seed at 50 snarls only gets $F1 = 0.98$ because a planted allele falls within the floor and both rules miss it). The anchor-interval rule gets the same result as the boundary-key rule up to 10^5 snarls with 1-bp boundaries (precision 1.000, $F1 \geq 0.998$) at no extra cost: both rules take 1.8-2.0s single-threaded at 10^5 snarls including pairing. This shows accuracy at 10^5 snarls, which is one to two orders of magnitude below the deployment scale of 10^6-10^7. The scaling runs go up to 10^9 nodes but only test property (i), which both rules pass. The number of boundary keys and the amount of drift increase as the graph gets larger (Fig. 4). The method fails only when two things happen: keys collide, and one graph has a snarl. That private snarl shifts every following snarl with the same key by one position, so a single missing snarl mispairs a chain of successors, and each of those mispaired snarls sends its alleles to both deltas. The anchor-interval rule avoids this problem because it restarts the alignment at every anchor.

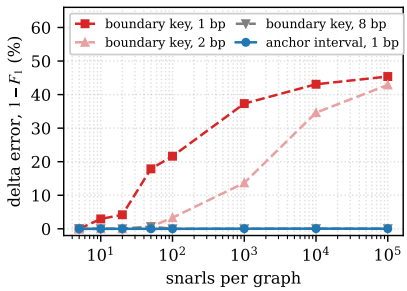


Fig. 4. Snarl-pairing error versus graph size, with 10% of the snarls private to one graph.

Real `vg construct` graphs. We built two HLA-DRB1 pangenomes from six public haplotypes (three per graph), aligned them with minimap2 and paftools, and constructed them with `vg construct` against GRCh38. The graphs contain 1552 and 1279 atomic variants: 1275 shared, 277 unique to A, and 4 unique to B. The diff recovers 273 of the 277 A-only variants (recall 0.99) at a precision of 0.78. In snarl-pairing mode, the boundary-key rule pairs 694 snarls and leaves 470 and 330 unpaired, as 10-13% of the keys are repeated on these graphs. The anchor-interval rule pairs 972, leaves 192 and 52 unpaired, and increases the alignment score by 53%. On real graphs with known answers, the reference-anchored diff recovers nearly every private variant, and its only false positives come from how variants are represented. When there are clusters of adjacent indels and SNPs, the diff may place a variant differently than the caller, so one variant can end up with two keys (§III-B4). All false positives are in such clusters, and two of the four B-only deletions are in a tandem repeat and are placed differently for the same reason.

Reference-anchored mode. On every planted workload, the reference-anchored mode achieves 100% precision and 100% recall, and a graph compared to itself produces an empty delta. The reference-anchored mode is exact when both graphs are identical, meaning they represent the same variant. Because snarls are matched by absolute coordinate instead of heuristic pairing, property (ii) is satisfied exactly.

B. Q2: Throughput and scaling

We measured on the 8-Mbp SV-like pair (a synthetic pair with SV-sized snarls and 25,608 snarl pairs). The DP phase achieves 2.0, 3.9, 7.9, 15.5, 29.8, and 51.9× at 2-64 cores (81% efficiency at 64), with results identical to the sequential run at every core count. End-to-end speedup levels off at 4.49× because the serial setup takes about 4.6 of the 22.1 s of a single-thread run. Popping several tasks per lock acquisition gives at most a 3% gain (53.4× for chunks of 4) and a loss for larger chunks (48.8 and 15.2× for 8 and 32, compared to 52.0× for single pops), since a chunk gives one thread several large tasks. The DP phase scales almost ideally up to 64 cores, but the end-to-end speedup levels off early (Fig. 5a). Three factors explain this gap: (1) the serial setup is not parallelized; (2) each snarl pair is a single, indivisible LAPJV call, so DP-phase speedup is limited to $W_{\text{total}}/W_{\max}$. With work scaling as k^3 and k reaching up to 40, the largest task is about 8,000 times heavier than a $k=2$ bubble (360 times in measured time, Fig. 3b). This heavy tail, which needs dynamic scheduling (Finding 2), also limits speedup; (3) all threads share a single mutex-guarded queue, and half the tasks are $k=2$ bubbles, so lock acquisition is not well amortized and contention may limit the benefit of more threads. We tested up to 64 cores and do not make claims beyond that.

We only time the baseline's extract-and-align stage (minimap2 runs with its default 3 threads, while PanG2G is single-threaded). We extract one haplotype per sample, while the graphs hold both, which favors the baseline. PanG2G's single-threaded time increases sub-linearly from 7.6 to 12.3 s as h goes from 2 to 200, while the baseline's time increases linearly from 7.0 to 624 s (the curves cross below h=5). Compared to linearize-and-map, PanG2G is 51 times faster at h=200, and the gap grows as h increases (Fig. 5b). This is because PanG2G's cost depends mainly on graph size, not the number of haplotypes. At h=2, the graph is still the full 51-Mbp chromosome-22 backbone, and adding more haplotypes just saturates the shared variation. The baseline, however, must expand the graph into h sequences and map each one. The baseline extracts haplotypes and maps them to the reference using minimap2 [18]. It produces a different output (per-haplotype alignments that still need calling and merging), and there is no other graph-to-graph tool for comparison. Mapping each haplotype of one population to the graph of the other using Giraffe [33] would remove reference bias on the mapping side, but would still require expanding one graph into h sequences, so its cost should also increase linearly with h. We have not benchmarked this.

C. Single-core scaling of the reference-anchored mode

We created synthetic pairs ranging from 8×10^6 to 8.2×10^9 bp, with the two graphs cutting the shared backbone at different points (25 vs. 17 bp) but both using the same set of planted variants and identical coordinate representations. PanG2G recovers the planted alleles exactly at every size. Wall time and peak memory scale as $N^{1.04}$ and $N^{0.99}$, and decomposition with the first-common-successor finder also

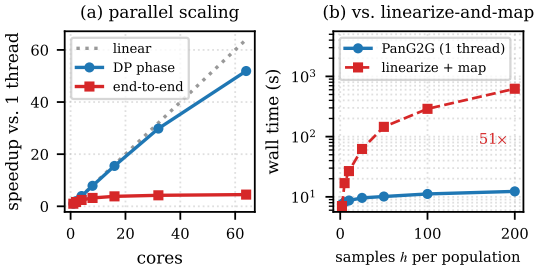


Fig. 5. (a) Parallel scaling to 64 cores (EPYC 7702) on the 8-Mbp SV-like pair: DP-phase speedup, and end-to-end speedup limited by the serial setup; the harness marks the run invalid if there is any divergence between the parallel and sequential results. (b) PanG2G's whole diff (single-threaded) vs. the extract-and-align cost of linearize-and-map on the real chromosome-22 pangenomes as the panel size h increases.

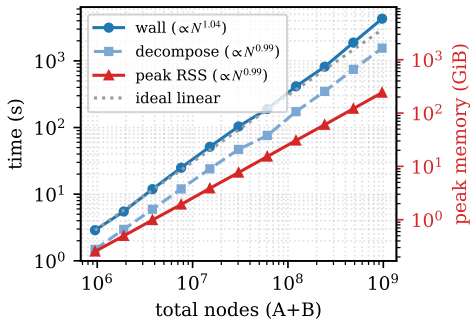


Fig. 6. Single-core scaling of the reference-anchored mode over 8×10^6 - 8.2×10^9 bp ($\sim10^6$-9.7×10^8 nodes): wall time $\propto N^{1.04}$ and decomposition $\propto N^{0.99}$ (left axis), peak memory $\propto N^{0.99}$ (right axis); dotted: ideal linear. Exact at every size.

scales as $N^{0.99}$. The largest pair (8.2×10^9 bp, 9.7×10^8 nodes, 4.1×10^7 variants; about 2.6× the human genome) is processed exactly in 72 minutes using 242 GiB peak memory. Throughput stays at 2.26×10^5 nodes/s at this size, with 270 GiB still available, so neither capacity nor bandwidth is a bottleneck. Using the superbubble scan, which reports the same snarls, decomposition at 4 Gbp (the largest size within the 192 GB job limit) takes 51 s instead of 745 and scales as $N^{1.07}$. Wall time drops from 1880 to 1475 s with memory unchanged, so the times above are conservative, and scaling then follows $N^{1.12}$ (the $N\log N$ of the variant maps). In reference-anchored mode, scaling is nearly linear with genome size and remains exact at all sizes (Fig. 6). The main computational cost comes from coordinate-keyed variant maps, which scale as $N\log N$, and from decomposition, which the superbubble scan makes near-linear.

D. Q3: Ablations

Exact LAPJV matching and the minimizer index are the main design choices. The scheduler only affects performance on skewed workloads, and the van Emde Boas layout has no effect at depth-1 (Fig. 7). We disabled each feature in turn (using one compile-time flag per binary) and measured the impact on two types of workloads: a control where the feature should not matter (balanced or L3-resident) and a treatment

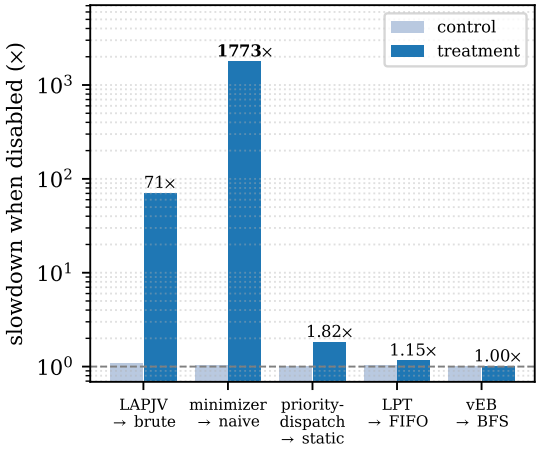


Fig. 7. Slowdown of the affected phase when each design choice is disabled, at a matched control and treatment workload (log scale).

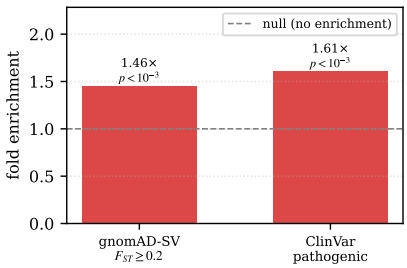


Fig. 8. PanG2G delta alleles on the real AFR vs. EAS pangenomes (1000 Genomes phase-3, chromosome 22) are enriched near both gnomAD-SV structural variants and ClinVar pathogenic sites (both $p < 10^{-3}$ against a permutation null; dashed line: no enrichment).

where it should (skewed, heavy-tailed, or large-k). LAPJV and the minimizer index are most important because brute-force matching becomes impractical as k increases (Finding 1), and the naive anchor scan is quadratic. Removing either increases their phase by 71× and 1773×, respectively, at the treatment scale. A static partition puts all heavy snarls on one thread (Finding 2), so a static cyclic partition adds a 1.82× on skewed work and 1.00× on balanced work. Downgrading LPT to FIFO adds another 1.15× once per-snarl work exceeds scheduling overhead. On depth-1 snarls, vEB order matches BFS order, so the layout remains 1.00× throughout. All ablated binaries return a delta that is byte-for-byte identical to the baseline.

E. Q4: Co-localization with structural and clinical variation

We built two population pangenomes using `vg construct` on GRCh37 chromosome 22 from the 1000 Genomes phase-3 panel (release 20130502): African (AFR, n=661) and East Asian (EAS, n=504). Comparing these in reference-anchored mode yields 608,718 delta alleles (428k unique to AFR, 181k unique to EAS). We then tested whether these alleles are more likely to be within 400 bp of a gnomAD-SV v2.1 structural variant [4] with AFR-EAS $F_{ST} \geq 0.2$ and a ClinVar pathogenic site [34] than expected by chance. We found 946 delta alleles near a gnomAD-SV (compared to 650 expected), which is significant (exceeds all 1000 permutations) and confirmed by synthetic positive and negative controls. The delta is more common in regions with structural and clinical variation: 1.46× higher near gnomAD-SV structural variants and 1.61× higher near ClinVar pathogenic sites (both $p < 10^{-3}$, (Fig. 8)). Population-specific alleles are found in areas with high variant density, which also include structural and clinically annotated variants.

However, this enrichment is not specific to population-differentiated SVs. The reason lies in the input: `vg construct` only includes what the VCF provides, and the VCF mainly contains SNPs and small indels, so the pangenomes are SV-depleted by design. Raising the F_{st} threshold from 0.05 to 0.5 does not change the fold (1.42-1.54×, all $p < 10^{-3}$), so the chromosome-22 delta overlaps with SVs in general, not just the most differentiated ones. Only 25 delta alleles insert or delete at least 50 bp, which is too few to test individually. Using an SV-containing panel, a frequency-aware delta, or a region-matched null could help isolate the differentiation, and the same pipeline would support all three. This experiment answers the question of "why pangenomes?": the graph-to-graph delta directly reveals real structural and clinical variation without needing a reference projection step.

V. Discussion

Each key design decision in PanG2G was made to address a workload property we measured. The snarl tree breaks down an NP-hard problem into smaller assignments. LAPJV, priority dispatch, and the flat layout each answer one of the three profiling findings, as the ablations confirm. The minimizer index and superbubble scan eliminate two previously quadratic stages, resulting in a pipeline that is near-linear in graph size.

A. Limitations

PanG2G is limited to acyclic graphs. Even if `vg snarls` provides the boundaries, allele enumeration and dynamic programming both assume a DAG, so tandem repeats are not supported. Our evaluation is limited to depth-1 and to graphs from the same builder. We have not tested nested Minigraph-Cactus graphs or cross-builder comparisons, which would require more complex normalization than simple left-alignment. The HLA experiment shows that even graphs from the same builder, when diffed by coordinate, inherit the caller's indel placement. Snarl pairing remains heuristic. The boundary-key rule can drift when keys repeat and snarls are unique to one graph. The anchor-interval rule removes this drift on our planted graphs but is only optimal within an interval. A full tree-matching pairing is the correct endpoint. Scaling beyond 64 cores, minimizer collisions beyond 10^8 k-mers and snarls with more than 2^{16} alleles have not been tested. PanG2G currently runs on a single machine and compares two graphs. Aligning more than two graphs, developing a distributed engine, and adding GPU acceleration for per-snarl matching are planned next steps.

VI. Conclusion

We introduced PanG2G, the first system that directly aligns two pangenome variation graphs at base-pair resolution. Its reference-anchored diff is the most rigorously validated output: it is exact whenever both graphs represent variants identically and scales almost linearly to an 8-Gbp pair on a single core. The parallel snarl-pairing aligner uses the full system design, including the minimizer index, LAPJV matching, priority-dispatch scheduler and vEB layout. We validated it on depth-1 workloads with up to 10^5 snarls. It achieves a $51.9\times$ DP-phase speedup on 64 cores, is 51 times faster than linearize-and-map at h=200, and with the anchor-interval rule it no longer drifts the way the boundary-key rule does. On real African and East Asian pangenomes, the delta co-localizes with known structural and clinical variation. Graph-to-graph alignment is now a practical tool for the pangenome era. The reference-anchored diff demonstrates its effectiveness today, and the parallel engine is a first step toward handling the general case.

a) Availability: The source code, Python prototype, synthetic generators (with the seeds used here), experiment harnesses, and measured result tables are all available at https://github.com/accl-kaust/pang2g. All real inputs are public: the 1000 Genomes phase-3 panel (release 20130502), GRCh37 and GRCh38, gnomAD-SV v2.1 sites [4], the ClinVar GRCh37 VCF [34], and the PGGB HLA-DRB1 test haplotypes.

b) Acknowledgment: The authors used Claude (Anthropic) to check spelling and grammar, help with phrasing throughout the manuscript, and generate Figs. 1 and 2; they reviewed and edited all AI-assisted content and take full responsibility for the article. The authors declare no conflict of interest.

References

[1] R. M. Sherman, J. Forman, V. Antonescu *et al.*, "Assembly of a pan-genome from deep sequencing of 910 humans of African descent," *Nature Genetics*, vol. 51, no. 1, pp. 30–35, 2019.

[2] W.-W. Liao *et al.*, "A draft human pangenome reference," *Nature*, vol. 617, no. 7960, pp. 312–324, May 2023.

[3] M. Kulmanov *et al.*, "Phased genome assemblies and pangenome graphs of human populations of Japan and Saudi Arabia," *Scientific Data*, vol. 12, p. 1316, 2025.

[4] R. L. Collins *et al.*, "A structural variation reference for medical and population genetics," *Nature*, vol. 581, no. 7809, pp. 444–451, May 2020.

[5] J. M. Eizenga *et al.*, "Pangenome graphs," *Annual Review of Genomics and Human Genetics*, vol. 21, no. 1, pp. 139–162, 2020.

[6] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970.

[7] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981.

[8] Y. Cui *et al.*, "A survey of sequence-to-graph mapping algorithms in the pangenome era," *Genome Biology*, vol. 26, no. 1, p. 138, May 2025.

[9] M. Coggi *et al.*, "A novel methodology for a comprehensive analysis of genomic sequence-to-graph alignment tools," *IEEE Transactions on Computational Biology and Bioinformatics*, vol. 22, no. 1, pp. 228–239, 2025.

[10] D. S. Cali *et al.*, "Segram: a universal hardware accelerator for genomic sequence-to-graph and sequence-to-sequence mapping," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 638–655.

[11] A. Gupta and N. Nishimura, "The complexity of subgraph isomorphism for classes of partial k-trees," *Theoretical Computer Science*, vol. 164, no. 1, pp. 287–298, 1996.

[12] B. Paten *et al.*, "Superbubbles, ultrabubbles and cacti," in *Research in Computational Molecular Biology*, S. C. Sahinalp, Ed. Cham: Springer International Publishing, 2017, pp. 173–189.

[13] B. Brejová *et al.*, "Maximum-scoring path sets on pangenome graphs of constant treewidth," *Frontiers in Bioinformatics*, vol. Volume 4 - 2024, 2024.

[14] R. Jonker and A. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems," *Computing*, vol. 38, no. 4, pp. 325–340, Dec 1987.

[15] C. Lee, C. Grasso, and M. F. Sharlow, "Multiple sequence alignment using partial order graphs," *Bioinformatics*, vol. 18, no. 3, pp. 452–464, 2002.

[16] J. Kececioglu and D. Starrett, "Aligning alignments exactly," in *Proceedings of the 8th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, 2004, pp. 85–96.

[17] C. Jain, H. Zhang, Y. Gao, and S. Aluru, "On the complexity of sequence-to-graph alignment," *Journal of Computational Biology*, vol. 27, no. 4, pp. 640–654, 2020.

[18] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 09 2018.

[19] E. Garrison *et al.*, "Variation graph toolkit improves read mapping by representing genetic variation in the reference," *Nature Biotechnology*, vol. 36, no. 9, pp. 875–879, Sep 2018.

[20] A. Banerjee *et al.*, "A work-optimal parallel algorithm for aligning sequences to genome graphs," in *2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2025, pp. 460–471.

[21] K. Riesen and H. Bunke, "Approximate graph edit distance computation by means of bipartite graph matching," *Image and Vision Computing*, vol. 27, no. 7, pp. 950–959, 2009.

[22] H. Li, X. Feng, and C. Chu, "The design and construction of reference pangenome graphs with minigraph," *Genome Biology*, vol. 21, no. 1, p. 265, Oct 2020.

[23] G. Hickey *et al.*, "Pangenome graph construction from genome alignments with minigraph-cactus," *Nature Biotechnology*, vol. 42, no. 4, pp. 663–673, Apr 2024.

[24] E. Garrison and A. Guarracino, "Unbiased pangenome graphs," *Bioinformatics*, vol. 39, no. 1, p. btac743, 01 2023.

[25] A. Guarracino, S. Heumos, S. Nahnsen, P. Prins, and E. Garrison, "ODGI: understanding pangenome graphs," *Bioinformatics*, vol. 38, no. 13, pp. 3319–3326, 2022.

[26] N. Noll *et al.*, "PanGraph: scalable bacterial pan-genome graph construction," *Microb. Genom.*, vol. 9, no. 6, Jun. 2023.

[27] S. Marcus *et al.*, "Splitmem: a graphical algorithm for pan-genome analysis with suffix skips," *Bioinformatics*, vol. 30, no. 24, pp. 3476–3483, 12 2014.

[28] E. Garrison *et al.*, "Pangenome graph variation format (PGVF) specification," https://github.com/pangenome/pgvf-spec, 2023.

[29] T. Onodera, K. Sadakane, and T. Shibuya, "Detecting superbubbles in assembly graphs," in *Algorithms in Bioinformatics (WABI 2013)*, ser. Lecture Notes in Computer Science, vol. 8126. Springer, 2013, pp. 338–348.

[30] P. van Emde Boas, "Preserving order in a forest in less than logarithmic time and linear space," *Information Processing Letters*, vol. 6, no. 3, pp. 80–82, 1977.

[31] M. Frigo *et al.*, "Cache-oblivious algorithms," in *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, 1999, pp. 285–297.

[32] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM J. Appl. Math.*, vol. 17, no. 2, p. 416–429, Mar. 1969.

[33] J. Sirén *et al.*, "Pangenomics enables genotyping of known structural variants in 5202 diverse genomes," *Science*, vol. 374, no. 6574, p. abg8871, Dec. 2021.

[34] M. J. Landrum *et al.*, "ClinVar: improving access to variant interpretations and supporting evidence," *Nucleic Acids Res.*, vol. 46, no. D1, pp. D1062–D1067, Jan. 2018.