Efficient Algorithms for Pangenome Personalization

Denys Andrukhovskyi
Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Slovakia

Martin Madzin
Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Slovakia

Luca Denti
Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Slovakia

Tomáš Vinař
Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Slovakia

Broňa Brejová
Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Slovakia

Abstract

A pangenome graph is a representation of the genomes of multiple individuals of the same species. Using a pangenome graph reference instead of a single linear reference genome can increase accuracy of read mapping and downstream tasks, e.g., variant calling, but can also lead to increasing computational demands and false positives. In 2024, Sirén et al. proposed to select only parts of the pangenome mostly likely to match a studied individual, introducing the so-called personalized pangenome reference. Their algorithm is based on greedily selecting sections of paths representing individual haplotypes comprising the pangenome. In this article, we formulate the problem of pangenome personalization purely in terms of pangenome vertices and edges, as finding two paths using vertices supported by sequencing data. We provide several algorithms for solving the problem, ranging from a simple linear-time greedy algorithm with approximation ratio analysis, through dynamic programming and application of minimum-cost flow. Our implementation misses only a small percentage of vertices belonging to the studied individual and improves the sensitivity of read mapping compared to the linear reference.

2012 ACM Subject Classification Applied computing → Computational genomics; Theory of computation → Design and analysis of algorithms

Keywords and phrases pangenome graph, approximation algorithm, network flow

Digital Object Identifier 10.4230/LIPIcs.WABI.2026.8

Supplementary Material *Software (source code)*: `https://github.com/fmfi-compbio/2paths` archived at `swh:1:dir:7b3fbbcb9b5bff8245a6e1d8f001bdcf50c93523`

Funding *Luca Denti*: This research work has received funding from the European Union's Horizon programme under the Horizon Europe grant agreement (ASVA-CGR No. 101180581).
Tomáš Vinař: This research was supported in part by the Slovak Grant Agency VEGA grant 1/0567/26.
Broňa Brejová: This research was supported in part by the Slovak Grant Agency VEGA grant 1/0140/25.

1 Introduction

A *pangenome* is a representation of a collection of genomes of multiple individuals of a species [11]. Although pangenomes can be easily described as a set of strings, each one representing the haplotype of an individual of the considered population [19], they are typically represented more compactly by means of graphs [20]. In a *pangenome graph*, vertices correspond to segments of DNA sequence and edges represent adjacency relationships

between these segments. A pangenome graph data structure can optionally also store the full *haplotype paths* representing the known haplotypes of the represented individuals. Pangenome graphs have been shown to decrease reference bias and improve accuracy and sensitivity of various bioinformatics tasks, such as read mapping [13, 25, 29], variant calling [10, 3, 2], and transcriptomic data analyses [26, 7].

Addressing human genome specifically, several large pangenome graphs containing many individuals were built [30, 20, 12], with the largest representing more than 1000 individuals [33]. However, larger graphs do not necessarily yield better downstream results [24, 28, 16, 6]. Indeed, the adoption of graphs containing many individuals reduces read mapping accuracy, hinders downstream analyses, and also introduces new computational challenges, such as increased memory and time requirements. To address these challenges, the idea of *personalized pangenome references* has been proposed [27, 31]. In a personalized pangenome reference, only a specific subset of the graph is selected based on the individual under investigation and downstream analyses are therefore performed using this smaller reference graph. In other words, personalizing a pangenome graph means simplifying its topology by retaining only known variants likely shared by the individual under analysis, thereby removing the additional complexity introduced by unshared and rarer variations. This approach has been shown to improve the accuracy of read mapping and variant calling, while also reducing the computational requirements [27].

Sirén et al. [27] proposed to personalize a pangenome by selecting several haplotypes (typically between 4 and 32) that best match the k-mers in the sequencing reads of the query individual. Each k-mer unique in the input pangenome is first classified as absent, heterozygous, or homozygous in the query individual based on its count in the input reads. The algorithm is a heuristic that repeatedly samples from existing haplotype paths in the pangenome, aiming for the first path to contain as many homozygous k-mers as possible, and subsequent paths to cover as many heterozygous k-mers as possible, while avoiding absent k-mers. The input pangenome graph is decomposed into shorter regions, and the algorithm is applied to each region, selecting haplotype paths in each region independently. If the same haplotype path is selected in an adjacent region, the two paths are merged into a single path; other haplotypes are combined into paths arbitrarily. This approach provides a practical solution for personalizing pangenome references, but it has several limitations. First, it relies on the presence of haplotype paths in the input pangenome graph, which may not always be available, especially in pangenomes constructed from fragmented assemblies. Second, the algorithm is a heuristic and does not define a proper optimization problem.

A related problem to pangenome personalization is the *haplotype path inference* problem [9, 5, 4]. The goal of this problem is to select one (in the haploid version) or two (in the diploid version) paths that best match the input reads while minimizing the number of recombinations between known paths. However, both versions of this problem are NP-hard and the algorithms proposed strictly require the haplotype paths to be present in the input graph.

In this work, we study the optimization problem of selecting two paths best matching sequencing reads from a query individual directly from a pangenome graph without considering haplotype paths. This is appropriate when haplotype paths are not available, e.g., in pangenome graphs constructed from fragmented assemblies or pangenomes from which low frequency vertices have been filtered out, fragmenting full-length haplotype paths. In our formulation, we consider directed acyclic graphs (DAGs) representing a pangenome graph consisting of a single connected component (e.g., a single chromosome). We assign a score to each vertex of the graph which depends on the number of times the vertex is used in the paths and then select two paths in the graph that maximize the overall score (Section 2).

We provide both exact and approximate algorithms for solving this problem (Section 3), and we demonstrate their performance on real pangenome graphs built from the variant call sets provided by the Human Pangenome Reference Consortium [20] and simulated read samples (Section 4). For our experiments, we use a simple scoring scheme based on the number of k-mers matching vertices of the pangenome graph, similar to scoring schemes used previously by others [9, 10].

2 Problem Definition

In this article, we consider a pangenome in the form of a directed acyclic graph $G = (V, E)$ with a single source vertex s and target vertex t. Each vertex is labeled with a string over alphabet $\{A, C, G, T\}$. Unless specified otherwise, a *path* refers to a directed path from s to t in G. Each such path represents a potential genomic sequence obtained by concatenating the labels of vertices on the path.

Our goal will be to select two (not necessarily distinct) paths in G containing the variants occurring on the two haplotypes of a single individual. A vertex $u \in V$ can be included in these paths in total 0, 1, or 2 times. We define a scoring function $w(i, u)$ for $i \in \{1, 2\}$, which can either assign a reward (positive score) or a penalty (negative score) for the path passing through the vertex. The first pass of vertex u invokes score $w(1, u)$, while the second pass of the same vertex invokes (possibly different) score $w(2, u)$.

We define the total score for vertex u as $T(0, u) = 0$, $T(1, u) = w(1, u)$, and $T(2, u) = w(1, u) + w(2, u)$. For two paths π_1, π_2, represented as sequences of vertices, the score is defined as:

$$S(\pi_1, \pi_2) = \sum_{u \in V} T(|\{i : u \in \pi_i\}|, u).$$

Thus, we can formulate our *2-paths problem* as follows: given a graph G, vertices s and t, and scoring function w, select paths π_1, π_2 maximizing score $S(\pi_1, \pi_2)$.

Some of our algorithms work for arbitrary scoring functions, while others put some restrictions on scores. In general, vertices supported by sequencing data will have positive scores, while those without support negative scores. One option of constructing the scoring function is to assign scores $w(1, u)$ based on agreement with sequencing data and then compute $w(2, u)$ in a unified fashion from $w(1, u)$. In our experiments, vertices with non-negative value of $w(1, u)$ get the score $w(2, u) = \alpha \cdot w(1, u)$ for some discounting factor $\alpha \in [0, 1]$. For $\alpha = 1$, this reduces to finding the highest scoring path using scores $w(1, u)$ and then using it twice, which does not correspond well to our goal of finding two haplotypes. For $\alpha < 1$, the second pass through a vertex will contribute less to the total score than the first pass. As a result, the second path is more likely to use other well-supported vertices and cover more of the variants present in the studied individual. For negative scores $w(1, u)$, the intuition is less clear. We decided to penalize the second pass more than the first, and thus we assign score $w(2, u) = (2 - \alpha)w(1, u)$. Note that for $\alpha = 1$, we have $w(2, u) = w(1, u)$ for both positive and negative scores, and thus the optimum solution (or one of several possible ones) will use the same path twice.

3 Algorithms

In this section, we introduce three algorithms to solve the 2-paths problem. We start with a simple greedy heuristic, which selects one path at a time. We show that it is not optimal and provide tight bounds on its approximation ratio (Section 3.1). The advantage of the

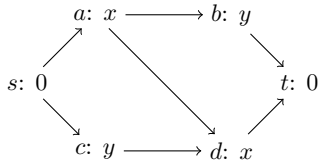

Figure 1 An example of a graph where the greedy algorithm may produce a suboptimal solution. Each vertex is represented by its identifier and value $w(1, u)$. We set $w(2, u) = \alpha \cdot w(1, u)$ for some $\alpha \in [0, 1)$. If $x > y > \alpha x$, the greedy algorithm selects path (s, a, d, t) and then either (s, a, b, t) or (s, c, d, t) for a total score of $2x + \alpha x + y$. The optimal solution uses paths (s, a, b, t) and (s, c, d, t) with a higher total score of $2x + 2y$. For example for values $x = 6, y = 5, \alpha = 1/2$, the greedy score is 20 and optimal 22.

greedy heuristic is its simplicity and efficiency, as it runs in $O(|V| + |E|)$ time. Next, we show a dynamic programming algorithm (Section 3.2) that works for an arbitrary scoring function, but its running time $O(|V| \cdot |E|)$ can be prohibitive. Fortunately, we can significantly reduce the running time by decomposing the graph into components that can be solved independently. Finally, we show a reduction to the minimum cost flow problem (Section 3.3), which can be solved in almost-linear time and also has efficient practical implementations. However, this reduction requires the scoring function to satisfy certain conditions.

3.1 Greedy Approximation Algorithm

The greedy algorithm we designed works as follows. We first score all vertices with $w(1, u)$ and find the longest weighted path π_1 in this graph using a simple dynamic programming in $O(|V| + |E|)$ time [8]. In the second step, we update the scores of vertices $u \in \pi_1$ to $w(2, u)$ and find the longest weighted path π_2 in this updated graph. The sum of weights of these paths is $S(\pi_1, \pi_2)$, but the two paths may not be optimal, as shown in Figure 1.

To improve this greedy heuristic, we could consider an iterative version, where after choosing π_2, we set $\pi_1 = \pi_2$, score the graph so that the vertices used in the new π_1 have score $w(2, u)$ whereas the remaining vertices have score $w(1, u)$, and then find the new π_2 as the longest weighted path with this scoring. Such iterations can be done repeatedly until there is no further improvement in the overall score.

The greedy algorithm can be applied on graphs with any scoring function, but we will now analyze its behaviour on graphs with non-negative scores that do not reward the second pass through a vertex more than the first. On such graphs, even the basic (non-iterative) greedy algorithm has a constant approximation ratio.

▷ Claim 1. Consider a constant $\alpha \in [0, 1]$ and inputs with a weight function such that $0 \leq w(2, u) \leq w(1, u)$ and $w(2, u) \geq \alpha \cdot w(1, u)$ for all $u \in V$. The greedy algorithm has an approximation ratio of at most $2/(1 + \alpha)$.

Note that the condition $w(2, u) \geq \alpha \cdot w(1, u)$ in the claim is trivially satisfied for $\alpha = 0$, and in that case the approximation ratio is 2.

Proof. Let π be the longest weighted path in the graph scored by $w(1, u)$ and let p be its score. The score of the greedy solution will be at least $p + \alpha p$, because after rescoring the graph, path π has the score of at least αp and the greedy chooses π_2 with at least this score. On the other hand, the optimal solution has a score of at most $2p$ because its first path π_1 uses scores $w(1, u)$ and thus has a score of at most p, and the second path may use some vertices with lower scores $w(2, u)$, and therefore its score is also at most p.

The approximation ratio is therefore at most $2p/(p + \alpha p) = 2/(1 + \alpha)$. ◁

By a more detailed analysis, we can prove the matching upper and lower bounds on the approximation ratio of the greedy algorithm.

▶ **Theorem 2.** *Consider a constant $\alpha \in [0,1]$ and inputs with a weight function such that $0 \leq w(2,u) \leq w(1,u)$ and $w(2,u) \geq \alpha w(1,u)$ for all $u \in V$. The greedy algorithm has an approximation ratio $4/(3+\alpha)$ on such inputs.*

Proof. First, we will prove the **upper bound** on the approximation ratio. For any two paths π, π', let $S(\pi') = \sum_{u \in \pi'} w(1,u)$ be the score of π' if all its vertices are used only once, and $S_\pi(\pi') = \sum_{u \in \pi' \setminus \pi} w(1,u) + \sum_{u \in \pi \cap \pi'} w(2,u)$ be the contribution of π' if π' was used together with π. We will use this notation not only for full paths, but also for arbitrary sets of vertices.

For a set of vertices U, let $\Delta(U) = \sum_{u \in U} (w(1,u) - w(2,u))$. Since $\Delta(U)$ is the sum of non-negative terms, it is monotonic with respect to set inclusion, that is, $\Delta(U') \leq \Delta(U)$ for $U' \subseteq U$. For any paths π and π', we can obtain the following equality from definitions of individual scores:

$$S(\pi') = S_\pi(\pi') + \Delta(\pi \cap \pi') \tag{1}$$

From inequality $w(2,u) \geq \alpha w(1,u)$, we can get $w(1,u) - w(2,u) \leq (1-\alpha) w(1,u)$, and thus for any set $U \subseteq V$ we have

$$\Delta(U) \leq (1-\alpha) S(U). \tag{2}$$

For any three paths π, π_1 and π_2, we apply the inclusion–exclusion principle and monotonicity of Δ to obtain the following inequality:

$$\begin{aligned} \Delta(\pi \cap \pi_1) + \Delta(\pi \cap \pi_2) = \\ \Delta\left((\pi \cap \pi_1) \cup (\pi \cap \pi_2)\right) + \Delta\left((\pi \cap \pi_1) \cap (\pi \cap \pi_2)\right) = \\ \Delta((\pi_1 \cup \pi_2) \cap \pi) + \Delta(\pi_1 \cap \pi_2 \cap \pi) \leq \\ \Delta(\pi) + \Delta(\pi_1 \cap \pi_2 \cap \pi) \leq \\ \Delta(\pi) + \Delta(\pi_1 \cap \pi_2). \end{aligned} \tag{3}$$

Let (π_1, π_2) be the optimal solution and (π, π') be the greedy solution, where π is the longest path with weights $w(1,u)$, and π' is the longest path after updating the scores. As π is the longest path, we have $S(\pi_i) \leq S(\pi)$ for $i \in \{1,2\}$. As π' is the longest path after updating the scores, we have $S_\pi(\pi_i) \leq S_\pi(\pi')$ for $i \in \{1,2\}$. Using equation (1) for π and π_i we can bound $S(\pi_i)$ with $S_\pi(\pi')$ for $i \in \{1,2\}$:

$$S(\pi_i) = S_\pi(\pi_i) + \Delta(\pi \cap \pi_i) \leq S_\pi(\pi') + \Delta(\pi \cap \pi_i) \tag{4}$$

Combining formulas (1), (4), (3) and (2), we can bound the score of the optimal solution $\mathfrak{O}$ as follows

$$\begin{aligned} \mathfrak{O} = S(\pi_1) + S_{\pi_1}(\pi_2) = \\ S(\pi_1) + S(\pi_2) - \Delta(\pi_1 \cap \pi_2) \leq \\ 2S_\pi(\pi') + \Delta(\pi \cap \pi_1) + \Delta(\pi \cap \pi_2) - \Delta(\pi_1 \cap \pi_2) \leq \\ 2S_\pi(\pi') + \Delta(\pi) + \Delta(\pi_1 \cap \pi_2) - \Delta(\pi_1 \cap \pi_2) = \\ 2S_\pi(\pi') + \Delta(\pi) \leq \\ 2S_\pi(\pi') + (1-\alpha) S(\pi). \end{aligned} \tag{5}$$

Also, we will use the trivial bound from the previous claim, namely:

$$\mathfrak{O} \leq S(\pi_1) + S(\pi_2) \leq S(\pi) + S(\pi) = 2S(\pi). \tag{6}$$

The score of greedy solution is $\mathfrak{G} = S(\pi) + S_\pi(\pi')$. Now we use a linear combination of inequalities (5) and (6) to complete the proof of the upper bound.

$$2\mathfrak{O} + (1+\alpha)\mathfrak{O} \leq 2\left(2S_\pi(\pi') + (1-\alpha)S(\pi)\right) + (1+\alpha)(2S(\pi)) = 4(S_\pi(\pi') + S(\pi)) = 4\mathfrak{G}.$$

Therefore, the approximation ratio is $\frac{\mathfrak{O}}{\mathfrak{G}} \leq \frac{4}{3+\alpha}$.

To prove the **lower bound**, we can assume that $\alpha < 1$ because for $\alpha = 1$ we get an upper bound on the approximation ratio 1, which is clearly optimal. For $\alpha \in [0, 1)$, let us consider the input from Figure 1. We assume that $x > y > \alpha x \geq 0$. Under these assumptions, the cost of the optimal solution is $2x + 2y$ and the cost of greedy is $2x + \alpha x + y$. We will set $x = y(1+\varepsilon)$ for $\varepsilon > 0$. Inequality $y > \alpha x$ is trivially satisfied for $\alpha = 0$ and for $\alpha \in (0, 1)$ it holds as long as $\varepsilon < \frac{1}{\alpha} - 1$.

$$\frac{\mathfrak{O}}{\mathfrak{G}} = \frac{2x + 2y}{2x + \alpha x + y} = \frac{2y(1+\varepsilon) + 2y}{2y(1+\varepsilon) + \alpha y(1+\varepsilon) + y} = \frac{4 + 2\varepsilon}{3 + 2\varepsilon + \alpha + \alpha\varepsilon}.$$

By taking the limit $\varepsilon \to 0^+$, we get:

$$\lim_{\varepsilon \to 0^+} \frac{\mathfrak{O}}{\mathfrak{G}} = \frac{4}{3+\alpha}.$$ ◀

3.2 Dynamic Programming Algorithm

Now we present a dynamic programming algorithm that finds an optimal solution for any scoring function, allowing both positive and negative vertex scores without any restrictions on the relationship between $w(1, u)$ and $w(2, u)$.

Our algorithm is similar to the algorithm for finding the optimal bitonic tour credited to Jon L. Bentley [8] and the sandwich algorithm for de novo peptide sequencing [21] in the sense that it simultaneously considers and extends two paths from the starting vertex s.

More precisely, the subproblem used in our algorithm is indexed by a pair of vertices u and v where $A[u, v]$ is the score of the optimal pair of paths, both starting in s, the first ending in u and the second ending in v. The trivial case is $A[s, s] = T(2, s) = w(1, s) + w(2, s)$. The score of the overall optimal solution is found in $A[t, t]$. We will fix a topological order of the vertices, which we denote by $\preceq$. Note that $A[u, v] = A[v, u]$, and thus we will treat indices of A as an unordered pair. For vertices u and v satisfying $s \preceq u \prec v$, the following recurrence holds:

$$A[u, v] = \max_{v' \in V : (v', v) \in E} A[u, v'] + w(1, v). \tag{7}$$

The recurrence considers all possible edges entering vertex v: one of them must be used in the optimal solution. Note that both u and v' precede v in the topological order, and thus the two paths in the optimal solution of $A[u, v']$ do not use v. By extending the path ending in v' by edge (v, v'), we get two paths with score $A[u, v'] + w(1, v)$. It can also be easily seen that if the optimal solution for $A[u, v]$ uses edge (v, v'), then after removing this edge we must get an optimal solution for $A[u, v']$, because otherwise we could improve the paths by substituting the optimal solution. These observations imply the correctness of this recurrence.

The case $A[v, v]$ is solved by a similar recurrence which now uses $w(2, v)$, because vertex v is used exactly once in the optimal solution of $A[v', v]$ and twice in $A[v, v]$:

$$A[v, v] = \max_{v' \in V:(v',v)\in E} A[v', v] + w(2, v). \tag{8}$$

This dynamic programming algorithm can be implemented by iterating over all vertices v in the topological order, for each v iterating over $u \preceq v$ in the topological order and computing $A[u, v]$ using these recurrences. Computation for one v is proportional to $|V|$ times the indegree of v, leading to total time $O(|V| \cdot |E|)$ and memory $O(|V|^2)$.

Such running time would be clearly impractical for large pangenome graphs. However, both running time and memory can be significantly improved by considering only values $A[u, v]$ for which $u \preceq v$ and either $u = t$ or there is an edge $(u, w) \in E$ such that $v \preceq w$ (we say that u has an edge reaching beyond v). Let X be the set of pairs of vertices that satisfy this condition.

When computing $A[u, v]$ for $(u, v) \in X$ according to Eq. 7, we consider values of $A[u, v'] = A[v', u]$, where either $u \preceq v' \preceq v$ or $v' \preceq u \preceq v$. If $u \preceq v'$, clearly pair $(u, v') \in X$, since u has an edge reaching beyond v and $v' \preceq v$. In the other case, where $v' \preceq u$, pair $(v', u) \in X$, because edge $(v', v) \in E$ reaches beyond $u \preceq v$. Similar argument can be applied to computation according to Eq. 8. This implies that since $(t, t) \in X$, we only need to use pairs included in X to compute the final value of $A[t, t]$.

An *articulation* in the pangenome graph is a vertex such that its removal will increase the number of connected components in the undirected version of G. Articulations may arise, for example, from conserved portions of the genome present without variation in all studied individuals. Articulations split the undirected version of the graph into a tree of biconnected components. We are only interested in one path of the tree connecting components containing s and t. Each biconnected component along this path corresponds to a block of consecutive vertices in the topological order. For $(u, v) \in X$, both u and v must belong to the same biconnected component, because there are no edges connecting different components. Thus, the running time can be bound by $O(|E| \cdot M)$ where M is the maximum size of a biconnected component in G.

3.3 Reduction to Minimum-Cost Flow

Another possibility is to reduce the problem to finding a minimum-cost flow in a directed graph. In theory, this problem can be solved in almost-linear time [32], and also practical solutions and an efficient solver implementation are available [14, 23]. In this reduction, we assume that in our scoring scheme we have $w(2, v) < w(1, v)$ for all vertices v. In other words, if a vertex is supported by sequencing reads, taking this vertex a second time gives a lower reward, whereas if a vertex is in conflict with sequencing reads, the penalty for taking this vertex a second time is larger than the first time.

Given a directed acyclic pangenome graph $G = (V, E)$ with source s and target t, we replace each vertex $v \in V$ with a gadget of four vertices v_1, v_2, v_3, v_4 with edges (v_1, v_3) and (v_2, v_4). We call vertices v_1 and v_2 the input ports of vertex v and vertices v_3 and v_4 its output ports. If (u, v) is an edge of the original graph, we will create four edges, connecting both output ports of u to input ports of v. The cost of all of these new edges will be 0, while cost of edges (v_1, v_3) and (v_2, v_4) denoted as c_{v_1,v_3} and c_{v_2,v_4} respectively will be derived from the scores assigned to vertex v as $c_{v_1,v_3} = -w(1, v)$ and $c_{v_2,v_4} = -w(2, v)$. As a final step, we create a universal source s' with edges with cost 0 connecting to input ports of s and we also create a universal sink t' connecting output ports of t to t'. All edges in this network will have capacity 1 and our task is to find the minimum-cost network flow from s' to t' of size 2. The construction of the network is illustrated in Figure 2.

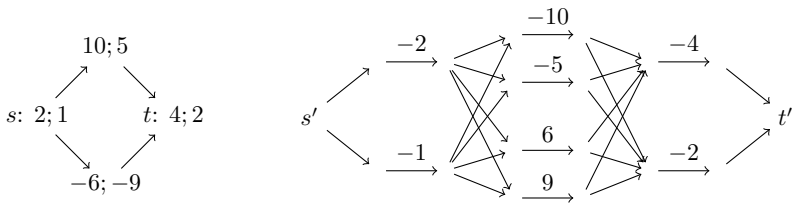

Figure 2 An example of a graph and the resulting flow network. Each vertex u in the graph on the left is labeled with $w(1, u)$ and $w(2, u)$ separated by a semicolon. The numbers in the flow network on the right show edge costs. Edges without a label have cost 0. All edges have capacity 1.

Intuitively, the resulting network flow will correspond to paths π_1 and π_2 that constitute a solution of our problem and the cost of the minimum cost flow will correspond to the score $-S(\pi_1, \pi_2)$. More precisely, if we consider two paths π_1 and π_2, we can create two edge-disjoint paths in our network, each path passing through gadgets corresponding to the vertices of the path in the original graph. If vertex v is used only once, the corresponding path in the network passes through vertices v_1 and v_3 incurring contribution to the final score of $-w(1, v)$. If vertex v is used twice, one of the network paths will pass through vertices v_1 and v_3 and the other one will pass through v_2 and v_4, incurring total cost of $-w(1, v) - w(2, v)$. Thus, the total cost of the corresponding network flow will sum up to $-S(\pi_1, \pi_2)$.

For the other direction, we use the integer property of minimum cost network flows: since all capacities in our network are 0 or 1 and the network flow is of size 2, there is a minimum cost flow that will use only flows of 0 or 1 on individual edges [1, Theorem 9.10]. This means that the flow can be decomposed into two edge-disjoint paths in the network, which translate to two paths π_1 and π_2 in the original graph. If vertex v is used twice, the incurred cost is clearly $-w(1, v) - w(2, v)$. If the vertex is used only once, instead, the corresponding path in the network must use edge (v_1, v_3), since the cost of that edge is smaller than cost of (v_2, v_4) (from the assumption $w(2, v) < w(1, v)$). In this case, the contribution to the total cost is $-w(1, v)$ and the minimum cost of the network flow corresponds to $-S(\pi_1, \pi_2)$. Therefore, the cost of a minimum cost network flow corresponds to the optimal solution of our problem and the paths π_1 and π_2 can be easily reconstructed from the solution of the minimum cost network flow problem.

Note that the argument can be extended to scoring schemes for which $w(2, v) \leq w(1, v)$ (as opposed to stricter $w(2, v) < w(1, v)$). However, in such cases, one needs to consider that gadgets for vertices where the equality holds may be traversed by multiple paths with equal cost.

4 Experiments

In this section, we describe the experiments that we performed to evaluate the methodology described in the previous section. The goal of our experimental evaluation was twofold: (i) to assess the correctness of our algorithm and compare the two versions proposed, namely the optimal solution obtained by the minimum cost flow solver and the greedy approximation, and (ii) to evaluate the effectiveness of our methodology in subsampling a pangenome graph.

To this end, we considered the `CHM13v2.0` reference genome [22] and built real pangenome graphs from the variant call sets (v1.1) provided by the Human Pangenome Reference Consortium [20] using the `vg` toolkit [13]. In our experiments, we restricted our analysis to

single chromosomes (namely, `chr1`, `chr13`, and `chr20`) and, for each chromosome considered, we built two pangenome graphs with different numbers n of individuals ($n \in \{17, 33\}$). Therefore, each graph contains $2 \times n + 1$ paths ($2 \times n$ haplotype paths and 1 reference path). We will refer to these graphs as `Full` graphs. We then selected a random individual (`HG01123`) and simulated a 20× paired-end Illumina sample from her haplotypes using `insilicoseq` [15]. Moreover, in order to simulate a real case scenario, where the sequenced individual is not actually already present in the pangenome graph, we removed her haplotype paths along with the private vertices and edges (i.e., those vertices and edges not shared with any other individual) from each `Full` graph, producing a new graph, which will be referred to as `1out` graph.

We then subsampled the graphs by solving the 2-paths problem using our implementation of the greedy and minimum-cost flow algorithms written in `C++`. The scoring of vertices for the 2-paths problem uses 31-mers computed using `KMC3` [17] from the simulated reads and is described in more detail in Section 4.1. Vertex scoring uses OpenMP for multithreading and the minimum-cost flow is solved by the OR-Tools library by Google [23]. To put our results in perspective, we compared our subsampling methodology to the subsampling implemented in the `vg` toolkit [27] (Sections 4.3 and 4.4).

4.1 Scoring Vertices

In our implementation, we use k-mer counts to assign scores $w(i, u)$ to vertices, comparing the k-mers represented by the pangenome graph with the k-mers present in the set of sequencing reads from the studied individual. Namely, let X be the set of k-mers (we use $k = 31$) that appear in the sequencing reads of the considered individual. To handle DNA strands, we use *canonical* k-mers, i.e., we compare a k-mer with its reverse complement and use the lexicographically smaller of the two for storing and querying the set X. We also omit from the set X the k-mers that appear only once in the set of reads, as these likely correspond to sequencing errors.

Let $\sigma(u)$ be the DNA sequence label of vertex u and for path $\pi = (u_1, \ldots, u_n)$, let $\sigma(\pi)$ be the concatenation of strings $\sigma(u_i)$, i.e. the genomic sequence represented by this path. The general idea of our scoring is to consider a multiset of k-mers X_u that are substrings of $\sigma(u)$ and assign a score $+1$ to each k-mer in X_u that also appears in X (which means it is supported by the reads) and score β to each k-mer in X_u that does not appear in X. The constant β should generally be zero or negative; in the experiments we use $\beta \in \{-0.5, -1.5\}$. The overall score $w(1, u)$ is then the sum of scores of individual k-mers from X_u.

However, this simple scheme would assign zero score to all vertices with $\sigma(u)$ shorter than k. Also, all k-mers spanning vertex boundaries would be completely ignored in scoring. To address these problems, we consider all paths starting in a particular vertex u. Given a path $\pi' = (u, u_1, u_2, \ldots)$, we extract the first $k - 1$ characters from $\sigma(u_1, u_2, \ldots)$, i.e., the first $k - 1$ nucleotides following $\sigma(u)$ in a potential genome. We append these $k - 1$ nucleotides to $\sigma(u)$ and score the k-mers in the resulting string. For a given u we might have multiple possible extensions of length $k - 1$. We score each of them and choose the highest score as $w(1, u)$.

In our scoring system, if the graph is a single path, every k-mer of the underlying string will be included in the score of exactly one vertex, namely, the vertex containing its starting position. In a more typical pangenome graph, many vertices have outdegree greater than 1. The score of a vertex then reflects the locally best path in its vicinity, but it is not guaranteed that this path is then indeed used in the final solution.

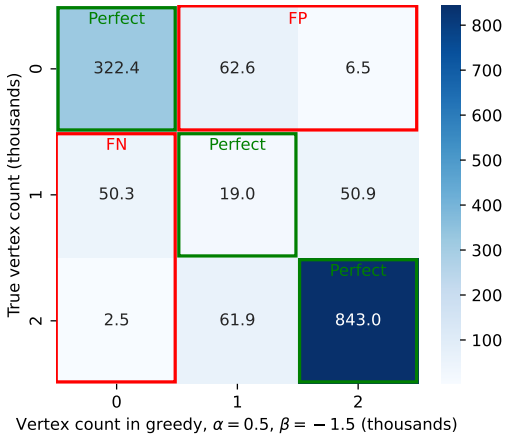

Figure 3 The confusion matrix comparing how many times was a vertex used in the true haplotypes and the paths selected by the greedy algorithm with $\alpha = 0.5$, $\beta = -1.5$. For each combination, we show the number of vertices in thousands. The matrix elements considered as "perfect", false positives (FP) and false negatives (FN) are shown by colored frames.

We compute the score for passing a vertex u a second time as follows: $w(2, u) = \alpha \cdot w(1, u)$ for $w(1, u) \geq 0$ and $w(2, u) = (2 - \alpha)w(1, u)$ for $w(1, u) < 0$. In the next section, we explore the influence of the discounting parameter α on the resulting pair of paths.

4.2 Recovery of True Vertices

In our first experiment, we ran our algorithms on the `Full` graph, which includes the vertices and edges unique to the studied individual. In this graph, we know the full haplotype paths corresponding to this individual. We will call these paths the true haplotypes. This experiment does not reflect real conditions, as when studying sequencing data from a new individual, we would not know the true haplotypes and parts of them will be missing from the graph. Nonetheless, this simple setting allows us to determine if we can recover vertices from the true haplotypes and to compare different values of parameters α and β as well as the greedy and optimal algorithms. The experiments were conducted on chromosome 20, using $n = 33$ individuals.

To evaluate the results, we consider for each vertex of the graph, how many times it occurs in the true haplotypes and how many times it occurs in the two paths predicted by our algorithm. This results in a confusion matrix of vertex counts for different combinations of vertex occurrences along the paths. An example of such a confusion matrix for the greedy algorithm, $\alpha = 0.5$ and $\beta = -1.5$ is shown in Figure 3. While incorrect typing of true homozygous loci is relatively rare, many vertices that should be part of heterozygous loci (true count 1) are apparently mistakenly classified as homozygous (predicted count 0 or 2). These errors likely happen in complex regions of the graph with many nearby variants that lead to many possible k-mer extensions, some of which may be by chance supported by read k-mers originating from a different part of the chromosome. This could be addressed by employing a more complex scoring scheme.

To compare different algorithms and parameter values, we summarize the correctness of our pair of paths by three values. The first one is the fraction of vertices that are located on the main diagonal of the confusion matrix, meaning that they have the same counts in the real haplotypes and in the predicted paths (see Figure 3). We will call these vertices as *perfectly predicted* vertices. A vertex is a *false positive* (FP) if it is not on either of the

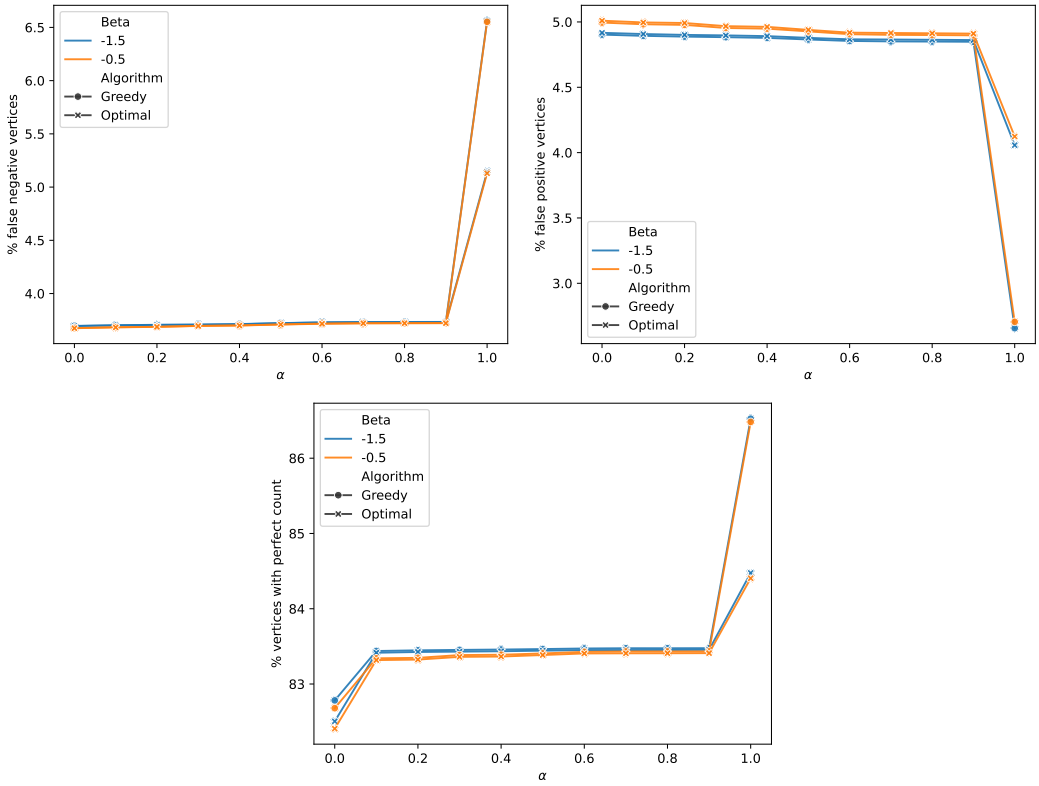

Figure 4 The ability of our algorithms to select correct vertices from the full graph. Top: The percentage of FN (left) and FP (right) vertices out of all vertices for two algorithms and different values of α and β (lower is better). Bottom: The percentage of perfectly predicted vertices out of all vertices for two algorithms and different values of α and β (higher is better).

real haplotypes but is at least on one of the predicted paths. Conversely, a vertex is a *false negative* (FN) if it is in at least on one of the real haplotypes, but is not on either of the predicted paths. For pangenome personalization, a false negative is generally worse than a false positive, because exclusion of a correct vertex prevents correct alignment of the overlapping reads, whereas inclusion of spurious vertices typically leads only to slight increase in time and memory of subsequent analyses, although in some cases such vertices may also introduce ambiguity in read mapping results.

In Figure 4, we can see that the false negatives are mostly below 4% and false positives mostly below 5% of all vertices. The rates of false positives and false negatives are relatively stable for values of α between 0 and 0.9, but change for $\alpha = 1$. At this setting, greedy selects the same path twice, which is optimal, but the flow-based approach selects two distinct paths with the same score, thus achieving a lower number of FNs but higher FPs. The percentage of vertices with perfect counts is again stable for values of α from 0.3 to 0.9 and reaches the highest value of 86.5% for the greedy algorithm with $\alpha = 1$ and $\beta = -1.5$. For $\alpha < 1$, all measures shown are only slightly influenced by the choice of the algorithm. Likewise, the change in β does not have a large influence.

Finally, Table 1 shows the approximation ratio of the greedy algorithm compared to the score obtained by the optimum flow-based algorithm, which is very close to 1 for all settings of α. We also computed the score of the true haplotypes in our scoring scheme, obtaining a considerably worse approximation ratio than for the greedy algorithm. This suggests that

Table 1 Approximation ratio of the greedy algorithm and the true haplotypes on the `Full` graph of chromosome 20 for 33 individuals. We show the ratio of the optimal score computed by minimum-cost flow to the score obtained by the greedy algorithm or the score obtained by scoring the paths belonging to the true haplotypes of the studied individual.

α	$\beta = -1.5$		$\beta = -0.5$	
	Greedy	True hapl.	Greedy	True hapl.
0.0	1+4e-06	1+0.02	1+3e-06	1+0.02
0.3	1+2e-06	1+0.01	1+2e-06	1+0.01
0.5	1+1e-06	1+0.01	1+9e-07	1+0.01
0.7	1+6e-07	1+0.01	1+5e-07	1+0.01
0.9	1+2e-07	1+0.01	1+1e-07	1+0.01
1.0	1	1+0.01	1	1+0.01

the errors caused by the inexact nature of the greedy approach are negligible compared to the problems caused by the relatively simple nature of our scoring scheme which is unable to distinguish all vertices originating from the true haplotypes from vertices irrelevant for the studied individual.

4.3 Read Mapping Evaluation

Finally, we applied our personalization approach to pangenome graphs constructed for three different chromosomes and two different numbers of individuals. For each input, we used the greedy algorithm with $\alpha = 0.5$ and $\beta = -1.5$. After computing the two paths, we use the `vg` toolkit [13] to index the pangenome consisting only of these paths and to map the simulated sequencing reads by `vg giraffe` [28]. We measure the number of reads that perfectly map to the graph across their full length and without any errors (see Figure 5). For comparison, we also map the same reads to four other references. The first reference is the `Full` graph, which includes the studied individual. Under ideal circumstances, all errorless reads should map perfectly to this graph. Although some perfect mappings may be missed due to various mapping heuristics, this input nonetheless represents a good upper bound of what we wish to achieve in the ideal case. The second reference is the `1out` graph, which excludes the studied individual, but the 16 or 32 individuals still capture many common variants present in the human population. This corresponds to a task of read mapping to a standard non-personalized pangenome. As we can see in Figure 5, the number of perfectly mapped reads compared to `Full` decreases only by 0.3-0.4%.

Next, we consider the personalized pangenome created by diploid sampling from `vg` [27]. Here the decrease compared to `Full` is in the range of 0.4-0.7%. Our method leads to somewhat fewer perfectly mapping reads, with the decrease compared to `Full` in the range of 1.5-2.2%. This lower mapping rate is due to the false negatives observed in Section 4.2. Another contributing factor is the fact that our method is unable to correctly phase the vertices into haplotypes, whereas `vg` diploid sampling creates the pangenome from long sections of real haplotype paths stored along the pangenome graph. Since `vg giraffe` relies on these haplotypes during read mapping, lack of phasing in our paths may cause some reads to be mapped incorrectly even if perfect path is present in the graph. Finally, we mapped the reads to the linear sequence of the corresponding chromosome from the human reference genome using `BWA-MEM` [18]. Here we observe the largest drop in the perfect mapping rate, in the range of 6.6-7.9%. This comparison reveals that our personalized genome still captures much of the variation present in the studied genome.

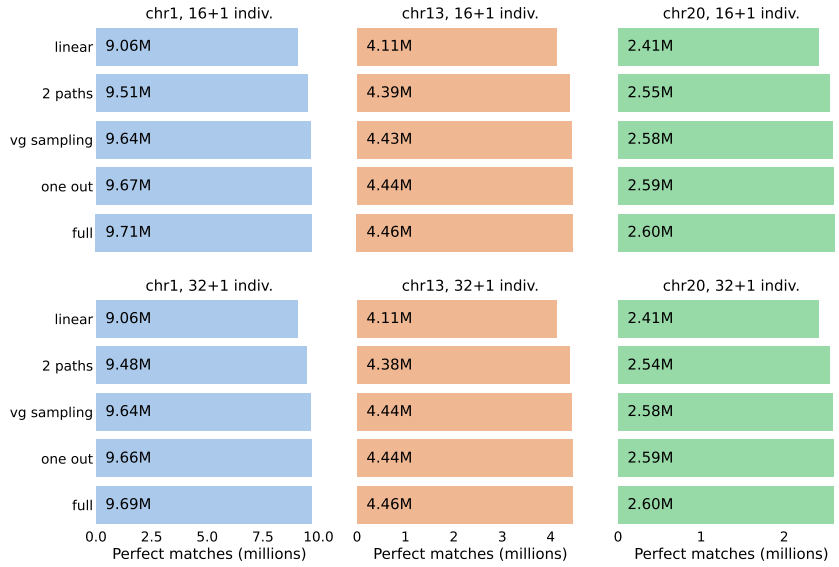

Figure 5 The number of perfectly matching reads for different references (in million reads). Plots for each chromosome use a different scale for easier comparison of methods within a chromosome.

4.4 Time and Memory Benchmarks

Finally, we compared the running times and memory requirements for the inputs and approaches considered in the previous experiment. We omit the `Full` pangenome as it is similar in size to the more realistic `1out` pangenome. All tools were allowed to use 16 threads. Figure 6 shows the overall running time split into phases, including k-mer counting, selecting 2 paths, indexing the resulting personalized pangenome and mapping the reads to it. The overall runtime of our approach is on most inputs slightly higher than `BWA-MEM` (which works on the linear reference) and slightly lower than mapping to the `1out` pangenome. The only exception is chromosome 13 for 16 individuals, where we observed increased running time of vertex scoring, perhaps due to some thread synchronization issues. However, it is worth noting that for our algorithm, graph indexing must be performed for every new individual. In contrast, the indexing step for `BWA-MEM` and `1out` can be reused for mapping reads of multiple individuals, since the reference remains the same. The mapping phase itself is only marginally faster for the personalized graph than for the `1out` pangenome. Specifically, for 32 individuals, the mapping is only 5-10% faster. However, perhaps bigger savings could be achieved on bigger or more diverse pangenomes. On the positive side, our personalization performs significantly faster than diploid sampling by `vg`.

In Figure 7, we see the memory consumption of each phase of all tested methods. For three pangenomes, the memory bottleneck is definitely the indexing phase. As expected, personalized pangenomes use smaller memory for indexing. The memory usage of mapping on the entire graph is comparable with the memory usage of the sampling phase of `vg` personalization. Our method uses less memory and scales better with graph size. For 32 individuals, personalized genomes use about 10-14% less memory compared to `1out`.

Overall, we conclude that time and memory requirements of our implementation are similar to other pangenome-processing tools. However, at this prototype stage, our tool is not able to produce significant time and memory savings to the overall process of read mapping, particularly if reads from multiple individuals need to be mapped. Perhaps more

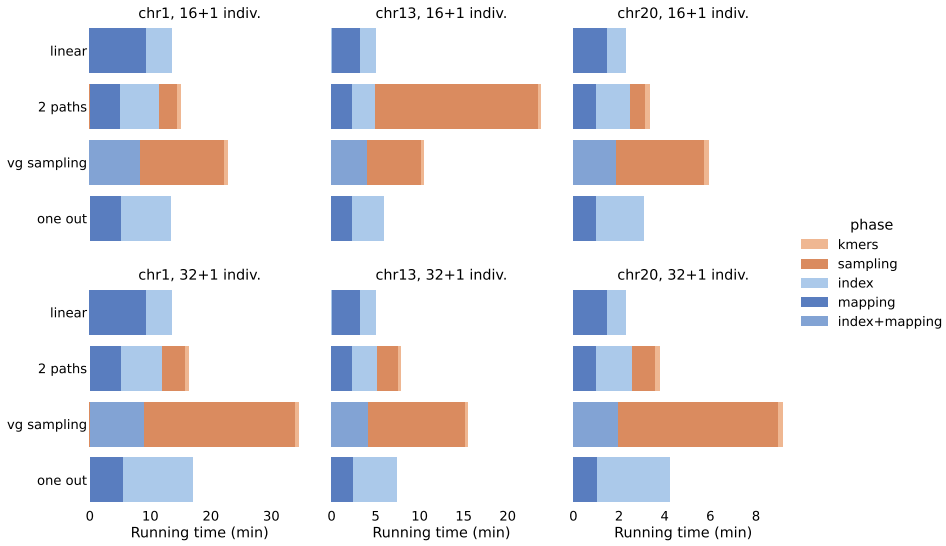

Figure 6 The running time for different references. Plots for each chromosome use a different scale for easier comparison of methods within a chromosome. For vg sampling, indexing and mapping were done in one step.

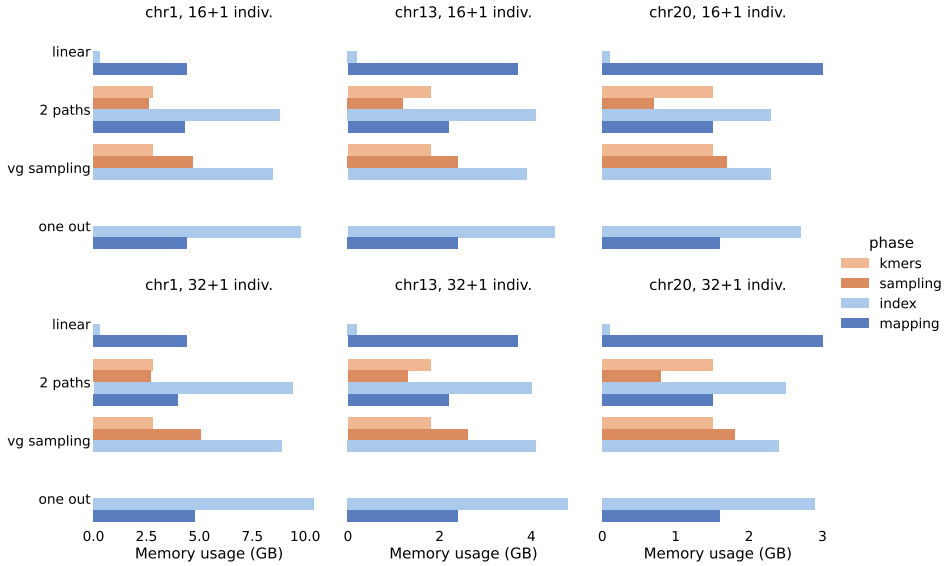

Figure 7 The memory for different references. Plots for each chromosome use a different scale for easier comparison of methods within a chromosome. For `vg` sampling, indexing and mapping were done in one step and are shown here as indexing.

significant savings could be achieved by tighter integration of personalization and mapping, such as producing index of the personalized pangenome by selecting portions of the index for the full pangenome.

5 Conclusion

In this paper, we introduced a new formulation of graph personalization as finding two paths in a directed acyclic graph with optimal score, where the score for passing a vertex a second time is discounted in some way, for example by a multiplicative factor α. We also presented three algorithms for solving this problem: a simple and efficient greedy algorithm with a provable approximation ratio, an exact dynamic programming algorithm, and a solution based on minimum-cost network flows. We implemented the first and third of these approaches and demonstrated that they can be successfully ran even on largest human chromosomes in reasonable amount of time and memory. The personalized pangenomes obtained by this method match the studied individual less well than `vg` personalization based on selecting long segments of known haplotypes. Nonetheless, they still produce a reference that matches the studied individual better than a single linear reference genome.

Advantages of our approach compared to the method implemented in `vg` are its cleaner combinatorial problem definition, which is easier to study theoretically, and applicability to graphs that do not have full haplotype paths available, such as those built from variants discovered by short-read sequencing. Our approach is also efficient: the greedy algorithm runs in linear time.

The fact that our method does not use haplotype paths is also its disadvantage in cases where these paths are available, leading to lower accuracy of our approach compared to `vg` haplotype sampling. Note that two paths recovered by our algorithm can by no means be considered as haplotypes, since we only consider the local structure of the underlying pangenome graph and make no attempt to preserve phasing.

Future work includes extending the problem beyond DAGs to at least some special classes of bidirected graphs often used to represent pangenomes. Also, extensions to finding more than two paths would be useful in situations with polyploid organisms or when "personalizing" the pangenome for a whole subpopulation.

Our approximation factor guarantees (Theorem 2) only cover the cases where all score values are non-negative. However, the scoring function actually used in the experiments assigns negative scores to unsupported vertices. Generally, it is unclear how to define approximation guarantees for scoring schemes with negative components. Perhaps, guarantees similar to Theorem 2 could be proved for cases where the final scores have high enough (positive) values. This hypothesis is empirically supported by our experiments which show that the results of the greedy algorithm are very close to optimal.

On a more practical side, better implementation and integration within `vg` or another pangenome toolkit would allow better practical applicability of our method. The method should also be tested on larger pangenomes and with real read sets. In a postprocessing step, the resulting paths could be locally phased in the areas with high density of heterozygous variants where multiple variants are spanned by input reads.

Finally, it might be beneficial to explore more complex scoring schemes. For example, tools for genome personalization or genome inference often classify k-mers as homozygous and heterozygous based on their counts in sequencing reads. This information could be incorporated to our scoring scheme so that the score for a homozygous k-mer would not be discounted during the second pass, while heterozygous k-mers would be discounted heavily.

References

1 R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

2 M. Asri, P. C. Chang, J. C. Mier, J. Siren, P. Eskandar, A. Kolesnikov, D. E. Cook, L. Brambrink, G. Hickey, A. M. Novak, L. Dorfman, D. R. Webster, A. Carroll, B. Paten, and K. Shafin. Pangenome-aware DeepVariant. *bioRxiv*, 2025. doi:10.1101/2025.06.05.657102.

3 S. Behera, S. Catreux, M. Rossi, S. Truong, Z. Huang, M. Ruehle, A. Visvanath, G. Parnaby, C. Roddey, V. Onuchic, A. Finocchio, D. L. Cameron, A. English, S. Mehtalia, J. Han, R. Mehio, and F. J. Sedlazeck. Comprehensive genome analysis and variant detection at scale using DRAGEN. *Nat Biotechnol*, 43(7):1177–1191, 2025.

4 G. Chandra, W. T. Doan, and D. Gibney. Haplotype-resolved diploid genome inference on pangenome graphs. *bioRxiv*, 2025. doi:10.1101/2025.11.26.690754.

5 G. Chandra, H. Helal, S. Scholz, A. T. Dilthey, D. Gibney, and C. Jain. Integer programming framework for pangenome-based genome inference. In *International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 405–408. Springer, 2025.

6 R. Chikhi, Y. Dufresne, and P. Medvedev. Constructing and personalizing population pangenome graphs. *Nat Methods*, 21(11):1980–1981, 2024.

7 S. Ciccolella, D. Cozzi, G. Della Vedova, S. N. Kuria, P. Bonizzoni, and L. Denti. Differential quantification of alternative splicing events on spliced pangenome graphs. *PLOS Computational Biology*, 20(12):e1012665, 2024.

8 T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

9 A. Dilthey, C. Cox, Z. Iqbal, M. R. Nelson, and G. McVean. Improved genome inference in the MHC using a population reference graph. *Nature Genetics*, 47(6):682–688, 2015.

10 J. Ebler, P. Ebert, W. E. Clarke, T. Rausch, P. A. Audano, T. Houwaart, Y. Mao, J. O. Korbel, E. E. Eichler, M. C. Zody, A. T. Dilthey, and T. Marschall. Pangenome-based genome inference allows efficient and accurate genotyping across a wide spectrum of variant classes. *Nature Genetics*, 54(4):518–525, 2022.

11 J. M. Eizenga, A. M. Novak, J. A. Sibbesen, S. Heumos, A. Ghaffaari, G. Hickey, X. Chang, J. D. Seaman, R. Rounthwaite, J. Ebler, M. Rautiainen, S. Garg, B. Paten, T. Marschall, J. Siren, and E. Garrison. Pangenome Graphs. *Annu Rev Genomics Hum Genet*, 21:139–162, 2020.

12 Y. Gao, X. Yang, H. Chen, X. Tan, Z. Yang, L. Deng, B. Wang, S. Kong, S. Li, Y. Cui, C. Lei, Y. Wang, Y. Pan, S. Ma, H. Sun, X. Zhao, Y. Shi, Z. Yang, D. Wu, S. Wu, X. Zhao, B. Shi, L. Jin, Z. Hu, Y. Lu, J. Chu, K. Ye, and S. Xu. A pangenome reference of 36 Chinese populations. *Nature*, 619(7968):112–121, 2023.

13 Erik Garrison, Jouni Sirén, Adam M. Novak, Glenn Hickey, Jordan M. Eizenga, Eric T. Dawson, William Jones, Shilpa Garg, Charles Markello, Michael F. Lin, Benedict Paten, and Richard Durbin. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature Biotechnology*, 36(9):875–879, 2018.

14 A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research*, 15(3):430–466, 1990. doi:10.1287/MOOR.15.3.430.

15 H. Gourlé, O. Karlsson-Lindsjö, J. Hayer, and E. Bongcam-Rudloff. Simulating Illumina metagenomic data with InSilicoSeq. *Bioinformatics*, 35(3):521–522, 2019. doi:10.1093/BIOINFORMATICS/BTY630.

16 C. Jain, N. Tavakoli, and S. Aluru. A variant selection framework for genome graphs. *Bioinformatics*, 37(S1):i460–i467, 2021.

17 M. Kokot, M. Długosz, and S. Deorowicz. KMC 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 2017. doi:10.1093/BIOINFORMATICS/BTX304.

18 H. Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv*, 2013. doi:10.48550/arXiv.1303.3997.

19 H. Li. BWT construction and search at the terabase scale. *Bioinformatics*, 40(12):btae717, 2024. doi:10.1093/BIOINFORMATICS/BTAE717.

20 W. W. Liao, M. Asri, J. Ebler, et al. A draft human pangenome reference. *Nature*, 617(7960):312–324, 2023.

21 B. Ma, K. Zhang, and C. Liang. An effective algorithm for peptide de novo sequencing from MS/MS spectra. *Journal of Computer and System Sciences*, 70(3):418–430, 2005. doi:10.1016/J.JCSS.2004.12.001.

22 S. Nurk, S. Koren, A. Rhie, M. Rautiainen, A. V. Bzikadze, A. Mikheenko, M. R. Vollger, N. Altemose, L. Uralsky, A. Gershman, et al. The complete sequence of a human genome. *Science*, 376(6588):44–53, 2022.

23 L. Perron and V. Furnon. OR-Tools v9.15, 2026. URL: https://developers.google.com/optimization/.

24 J. Pritt, N. C. Chen, and B. Langmead. FORGe: prioritizing variants for graph genomes. *Genome Biol*, 19(1):220, 2018.

25 M. Rautiainen and T. Marschall. GraphAligner: rapid and versatile sequence-to-graph alignment. *Genome Biol*, 21(1):253, 2020.

26 J. A. Sibbesen, J. M. Eizenga, A. M. Novak, J. Sirén, X. Chang, E. Garrison, and B. Paten. Haplotype-aware pantranscriptome analyses using spliced pangenome graphs. *Nature Methods*, 20(2):239–247, 2023.

27 J. Sirén, P. Eskandar, M. T. Ungaro, et al. Personalized pangenome references. *Nature Methods*, 21(11):2017–2023, 2024.

28 J. Siren, E. Garrison, A. M. Novak, B. Paten, and R. Durbin. Haplotype-aware graph indexes. *Bioinformatics*, 36(2):400–407, 2020. doi:10.1093/BIOINFORMATICS/BTZ575.

29 J. Siren, J. Monlong, X. Chang, A. M. Novak, J. M. Eizenga, C. Markello, J. A. Sibbesen, G. Hickey, P. C. Chang, A. Carroll, N. Gupta, S. Gabriel, T. W. Blackwell, A. Ratan, K. D. Taylor, S. S. Rich, J. I. Rotter, D. Haussler, E. Garrison, and B. Paten. Pangenomics enables genotyping of known structural variants in 5202 diverse genomes. *Science*, 374(6574):abg8871, 2021.

30 H. S. Tetikol, D. Turgut, K. Narci, G. Budak, O. Kalay, E. Arslan, S. Demirkaya-Budak, A. Dolgoborodov, D. Kabakci-Zorlu, V. Semenyuk, A. Jain, and B. N. Davis-Dusenbery. Pan-African genome demonstrates how population-specific genome graphs improve high-throughput sequencing data analysis. *Nat Commun*, 13(1):4384, 2022.

31 K. Vaddadi, M. J. Lin, S. Majidian, T. Mun, and B. Langmead. Minimizing reference bias with an imputed personalized reference. *Genome Res*, 36(4):740–753, 2026.

32 J. van den Brand, L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. P. Gutenberg, S. Sachdeva, and A. Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 503–514. IEEE, 2023.

33 Y. Wang, Z. Duan, D. Chen, D. Shi, Y. Ding, Z. Wang, B. Li, Z. Wang, M. Guo, W. Yang, J. Hou, W. Chen, Y. Guo, W. Wei, Y. Cao, X. Sun, W. Bai, M. Lu, T. Qi, X. Shen, and J. Yang. The 1000 Chinese Pangenome empowers medical and population genetics. *Nature*, 654:121–130, 2026.