JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 25, Number 0, 2018
© Mary Ann Liebert, Inc.
Pp. 1–15
DOI: 10.1089/cmb.2017.0264

Research Article

Sequence Alignment on Directed Graphs

VADDADI NAGA SAI KAVYA, KSHITIJ TAYAL,
RAJGOPAL SRINIVASAN, and NAVEEN SIVADASAN

Downloaded by Univ Of Western Ontario from www.liebertpub.com at 09/12/18. For personal use only.

ABSTRACT

Genomic variations in a reference collection are naturally represented as genome variation graphs. Such graphs encode common subsequences as vertices and the variations are captured using additional vertices and directed edges. The resulting graphs are directed graphs possibly with cycles. Existing algorithms for aligning sequences on such graphs make use of partial order alignment (POA) techniques that work on directed acyclic graphs (DAGs). To achieve this, acyclic extensions of the input graphs are first constructed through expensive loop unrolling steps (DAGification). Furthermore, such graph extensions could have considerable blowup in their size and in the worst case the blow-up factor is proportional to the input sequence length. We provide a novel alignment algorithm V-ALIGN that aligns the input sequence directly on the input graph while avoiding such expensive DAGification steps. V-ALIGN is based on a novel dynamic programming (DP) formulation that allows gapped alignment directly on the input graph. It supports affine and linear gaps. We also propose refinements to V-ALIGN for better performance in practice. With the proposed refinements, the time to fill the DP table has linear dependence on the sizes of the sequence, the graph, and its feedback vertex set. We conducted experiments to compare the proposed algorithm against the existing POA-based techniques. We also performed alignment experiments on the genome variation graphs constructed from the 1000 Genomes data. For aligning short sequences, standard approaches restrict the expensive gapped alignment to small filtered subgraphs having high similarity to the input sequence. In such cases, the performance of V-ALIGN for gapped alignment on the filtered subgraph depends on the subgraph sizes.

Keywords: genome variation graphs, pangenome, sequence alignment, V-ALIGN.

1. INTRODUCTION

MOST STATE-OF-THE-ART HIGH-THROUGHPUT GENOME STUDIES rely heavily on mapping sequences to a high-quality reference genome (Lander et al., 2001). Use of a single reference sequence, however, has limited capability in representing significant genomic diversity and it suffers from reference allele bias during interpretations (Novak et al., 2017; The Computational Pan-Genomics Consortium, 2018). The number of sequenced genomes is, however, increasing and this is driving a paradigm shift in genome analysis from single reference sequence based to pangenome reference (Novak et al., 2017; VG, 2017; The Computational Pan-Genomics Consortium, 2018).

TCS Research, Hyderabad, India.

Representing genomic variations using graph data structures has recently attracted considerable interest (Paten et al., 2011a; Dilthey et al., 2015; Novak et al., 2017; VG, 2017; The Computational Pan-Genomics Consortium, 2018). Various graph data structures with subtle distinctions have been studied in the literature for pangenome representation (The Computational Pan-Genomics Consortium, 2018). These include De Bruijn graphs (De Bruijn, 1946; Compeau et al., 2011), A-Bruijn graphs (Pevzner et al., 2004), Enredo graphs (Paten et al., 2008), Cactus graphs (Paten et al., 2011a,b), Population Reference graphs (Dilthey et al., 2015), String graphs (Myers, 2005), and Variation graphs (Novak et al., 2017). The broad idea behind these representations is to effectively encode genomic variations such as insertions, deletions, duplications, and transpositions, as alternative paths in a directed graph. Such graph-based representations have shown promise in improved read mapping and variant calling performance (Novak et al., 2017). Moving to a graph-based reference has necessitated the development of graph-based computational pipelines for genome analyses (Novak et al., 2017; VG, 2017; The Computational Pan-Genomics Consortium, 2018).

Genome variation graphs provide a natural representation for pangenome (Novak et al., 2017; The Computational Pan-Genomics Consortium, 2018). Common subsequences are encoded as vertices and variations are captured using additional vertices and directed edges. The pangenome sequences are present as directed paths in these graphs. In the case of a single reference sequence, the graph is just a single vertex corresponding to the entire sequence. Complex structural variations could introduce directed cycles in these graphs. For example, the 1000 Genomes data show existence of many short tandem repeats (STRs) where the repeat units (RUs) have highly varying repeat counts across the genomes (STR, 2017). For instance, Chromosome 11 contains RU whose repeats per allele (RPA) values vary from 1 to 27. Encoding such variations could introduce cycles in the corresponding genome variation graph.

In this article, we consider the fundamental problem of sequence alignment. In particular, we consider the alignment of a sequence to a pangenome reference that is encoded as a genome variation graph. Our goal is to compute an alignment of the input sequence to a path in the graph having maximum alignment score among all paths. We consider gapped alignments where the gaps could be affine, linear, or constant. The formal problem definitions are given in Section 2.

An algorithm for aligning a new sequence to a multiple sequence alignment (MSA) encoded as graph was given in Lee et al. (2002). MSA is encoded as a partial order alignment (POA) graph and the alignment algorithm aligns the new sequence to the POA (Lee et al., 2002). The POA-based algorithm discussed in Lee et al. (2002) is an extension of the traditional dynamic programming (DP) algorithms for sequences (Smith and Waterman, 1981; Gotoh, 1982) to handle partial orders. POA graphs are directed acyclic graphs (DAGs) where the sequences are encoded as paths. Under such a formulation, gapped alignment of an m length sequence to a POA graph on E edges takes $O(mE)$ time (Lee et al., 2002).

The POA graphs share resemblance to genome variation graphs in the sense that the variations are encoded as alternative paths using additional vertices and edges. In Novak et al. (2017), POA-based technique was used for aligning sequences to genome variation graphs. POA-based techniques are restricted to acyclic graphs whereas genome variation graphs can have cycles. To handle cycles, Novak et al. (2017) first construct acyclic extensions of the input graphs through expensive loop unrolling (DAGification) steps and the alignment is then performed on the acyclic extensions. A k-length DAGification of a graph G aims to compute a DAG G' such that all paths (not necessarily simple) of length k or less in G are present in G' and vice versa. For aligning an m length sequence, the value of k has to be m or more.

Acyclic graph extensions can have considerable blowup in their size. The edge and vertex blow-up factor in the worst case is proportional to the input sequence length. Prohibitively large size of the DAGified graph results in increased preprocessing and alignment time and thereby affects the overall alignment performance. For large reference graphs, the sequence alignment follows a seed and extend strategy where candidate subgraphs of the reference graph with potentially large alignment scores are first identified (Novak et al., 2017). The final alignment is then performed on these filtered subgraphs. In this case, the DAGification is restricted to these candidate subgraphs.

In this article, we provide a novel alignment algorithm V-ALIGN that aligns the input sequence directly on the genome variation graph while avoiding the expensive DAGification preprocessing. It computes an alignment of the input sequence to a path in the graph having maximum alignment score among all paths. V-ALIGN is based on a novel DP formulation that allows gapped alignment with affine, linear, or constant gaps directly on the input graph. We also propose refinements to V-ALIGN for better performance in

Downloaded by Univ Of Western Ontario from www.liebertpub.com at 09/12/18. For personal use only.

practice. In this, the time to fill the DP table has linear dependence on the sizes of the sequence, the graph, and its feedback vertex set. A feedback vertex set of a graph is a subset of its vertices whose removal makes the graph acyclic. The runtime of this algorithm matches that of the POA-based technique when the graph is acyclic. V-ALIGN performs one-time preprocessing of the graph to compute pairwise edge distances between the vertices and to compute a feedback vertex set. When the alignment is restricted to a filtered set of subgraphs, which is done for improved performance when the graph is large, V-ALIGN can be used for aligning to these candidate subgraphs. In this case, its performance depends on the subgraph sizes. We provide a theoretical result on the complexity of the DAGification preprocessing that is required by the POA-based technique. We show that two-length DAGification of a graph G where the resultant DAG has minimum number of vertices is NP-complete. We additionally conduct empirical studies on the DAGification overhead. For this, we use a depth first search (dfs)-based DAGification algorithm and measure the blowup in the vertices and edges of the resultant graphs for different types of input graphs. We also perform alignment experiments on the genome variation graphs constructed from the 1000 Genomes data.

V-ALIGN is implemented in C++ and is freely available for download and use from https://github.com/tcsatc/valign. Details of V-ALIGN usage and reference graph format are discussed in Sections 1 and 5 of the Supplementary Material. The V-ALIGN tool supports easy visualization of the alignment by generating output that can be visualized using the standard Graphviz tools (Graphviz, 2017; V-ALIGN, 2017) (Section 6 of the Supplementary Material).

Downloaded by Univ Of Western Ontario from www.liebertpub.com at 09/12/18. For personal use only.

2. PRELIMINARIES

2.1. Notations

Let $G=(V, E, \gamma)$ be a connected directed graph with vertices V, edges E, and vertex labels given by $\gamma(v)$. Edges in E are represented as ordered pairs from $V \times V$. Let Σ^+ denote the set of all sequences of one or more elements from an alphabet Σ. For nucleotide sequences, Σ is the set of nucleotides. For a vertex $v \in V$, its label $\gamma(v) \in \Sigma^+$. A directed path p in G of length r vertices is denoted by the ordered sequence $(u_1, \ldots, u_r)$, where $u_i \in V$ and $(u_i, u_{i+1}) \in E$. We only consider paths with length >0. We say that the path p starts at u_1 and ends at u_r. Let $P(v)$ denote the set of all directed paths in G that end at vertex v. Clearly the cardinality of $P(v)$ could be infinity if there are directed cycles in G. For an ordered sequence $x=(x_1, x_2, \ldots, x_m)$, let $|x|$ denote the length of sequence x, which is m here. For a directed path $p=(u_1, \ldots, u_t)$ in G (not necessarily simple), we call the sequence obtained by concatenating $\gamma(u_1), \ldots, \gamma(u_t)$ in the same order as the label of the path p and is denoted by $\gamma(p)$. Hence $|\gamma(p)| = \sum_{i=1}^{t} |\gamma(u_i)|$. For any contiguous subsequence y of $\gamma(p)$, we say that G contains the label sequence y. For a vertex $v \in V$, let $I(v)$ called the in neighbors of v denote the set of all vertices that have directed edges to v. For a set E, we also use E to denote its cardinality in place of $|E|$ inside asymptotic notations for better readability.

2.2. Alignment problems

We consider gapped alignment of a sequence $x \in \Sigma^+$ to G, where the goal is to compute an alignment of x to $\gamma(p)$ for some path p in G, that achieves the maximum alignment score among all paths in G. Since G is a directed graph possibly with cycles, the standard global alignment and local alignment between sequences translate to the following two variants:

- End to end alignment of x to a contiguous subsequence y of $\gamma(p)$, without penalizing the unmatched suffix and prefix of $\gamma(p)$. The maximum alignment score is denoted as $g(x, \gamma(p))$.
- Local alignment of x and $\gamma(p)$. The maximum alignment score is denoted as $\ell(x, \gamma(p))$.

Consequently, we define

$$g(x, G) = \max_{\{paths\ p \in G\}} g(x, \gamma(p))$$

and

$$\ell(x, G) = \max_{\{paths\ p \in G\}} \ell(x, \gamma(p)).$$

3. METHODS

3.1. Basic V-ALIGN algorithm

For the ease of exposition, we assume that the vertex labels are length one sequences from Σ^+. That is, the label of any vertex in G is an element from Σ. We discuss later how the algorithm can be easily modified to handle the general case. In the following, we define a DP formulation that would allow us to find optimal alignments.

For vertices u and v in G, let $\delta(u, v)$ denote the minimum number of edges on any directed path from u to v in G. That is, $\delta(u, v)$ is the shortest edge distance from u to v. Clearly $\delta(u, u) = 0$. If there is no directed path from u to v, then $\delta(u, v) = +\infty$. For $a, b \in \Sigma$, let $s(a, b)$ denote the substitution score between a and b. Let $\Delta(k)$ denote the penalty for a k length gap, such as affine, linear, or constant gap. $\Delta(0) = 0$ by definition.

Let the input sequence be $x = (x_1, \ldots, x_m)$ of length m. We consider an arbitrary linear ordering of the vertices in V. Let M denote the scoring matrix of size $|V| \times (m+1)$, where $M(w, j)$ is the entry for $w \in V$ and $j \in [0, m]$. We use the following recurrence relation on $M(w, j)$ for all $j \in [1, m]$ and $w \in V$:

$$M(w, j) = \max \begin{cases} M(w, j-k) - \Delta(k) \quad \text{for all} k \in [1, j] \\ M(u, j-1) + s(\gamma(v), x_j) - \Delta(\delta(v, w)) \\ \qquad \text{for all} (u, v) \in E \\ 0 \quad \text{[for local alignment]} \end{cases}.$$

The entry $M(w, j)$ stores the maximum score for aligning the subsequence $(x_1, \ldots, x_j)$ to any path ending at vertex w in G. The first term of the mentioned max expression corresponds to an alignment having k gaps in the end due to the deletion of the last k elements of $(x_1, \ldots, x_j)$. The second term corresponds to aligning x_j to an intermediate vertex v in the path followed by gaps due to the deletion of the remaining path, whose length is no more than $\delta(v, w)$ for an optimal alignment.

There could be vertices in G with no incoming edges (0 in degree). To handle such vertices, we always include a dummy vertex θ in the vertex set V and add directed edges from θ to each vertex in G with 0 in degree. Matrix M is initialized as $M(w, 0) = 0$ for each $w \in V$ and $M(\theta, j) = 0$ for all $j \in [0, m]$ for local alignment. For the score function $g(x, G)$, the 0 term is absent from the max expression in the mentioned recurrence and M is initialized as $M(w, 0) = 0$ and $M(\theta, j) = -\Delta(j)$ for $j \in [1, m]$. Computing alignment score $\ell(x, G)$ and an alignment path from M are done in the usual manner. For $g(x, G)$, the alignment score is the largest $M(v, m)$ entry.

The computational efficiency can be improved further using standard techniques (Gotoh, 1982) (for affine, linear, or constant gaps) by defining an auxiliary matrix Q, where

$$Q(w, j) = \max\{M(w, j-k) - \Delta(k)\} \text{ for } k \in [1, j]$$

and replacing the first term in the max expression mentioned with $Q(w, j)$. Value of $Q(w, j)$ can be updated in $O(1)$ time because of the recurrence $Q(w, j) = \max\{M(w, j-1) - \Delta(1),\ Q(w, j-1) - t\}$, where t is the gap extension cost. $Q(w, 0) = -\infty$ for all w. The time complexity for filling M is $O(mVE)$. Computing $\delta(u, v)$ is a one-time preprocessing that can be done in $O(VE)$ time.

3.2. Improved V-ALIGN algorithm

We now provide a modified DP formulation to compute M that can achieve better run time performance in practice. Consider some linear ordering of the vertices in V. We can assume that the dummy vertex θ is the first vertex in the ordering. A vertex v is called in order with respect to this given ordering if all vertices in $I(v)$ (in neighbors of v) lie to the left of v in this ordering. Let $V' \subseteq V$ denote the set of all vertices that are not in order. We note that if G is acyclic (DAG), then the topological sorting gives an ordering where $V' = \phi$. In directed graphs with cycles, $|V'| > 0$. If G can be made acyclic (DAG) by removing at most α edges, then clearly $|V'| \leq \alpha$. This is because introducing all the deleted α edges to a topological sorted order of the DAG can make at most α vertices not in order.

We assume that the matrix rows are permuted with respect to the linear ordering of V. The technique in Gotoh (1982) for sequences can be extended to handle our case as follows. The earlier recurrence for $M(w, j)$ can be rewritten in the following manner. For $j \in [1, m]$ and for all w except θ,

Downloaded by Univ Of Western Ontario from www.liebertpub.com at 09/12/18. For personal use only.

$$M(w,j) = \max \begin{cases} M(u, j-1) + s(\gamma(w), x_j) \\ \qquad \text{for all } u \in I(w) \\ Q(w,j) \\ R(w,j) \\ 0 \quad \text{[for local alignment]} \end{cases},$$

where

$$R(w,j) = \max \begin{cases} M(u, j-1) + s(\gamma(v), x_j) - \Delta(\delta(v,w)) \\ \qquad \text{for all } (u,v) \in E \text{ where } v \neq w \end{cases}.$$

As earlier, Q can be computed efficiently using the recurrence

$$Q(w,j) = \max\{M(w, j-1) - \Delta(1),\ Q(w, j-1) - t\}.$$

We recall that the matrix rows are permuted with respect to the linear ordering of V. This ensures that while computing the matrix entry (w, j) for some vertex $w \in V - V'$, the values of the entries (u, j) for all $u \in I(v)$ (the in neighbors of v) are already available. Hence, if $w \in V - V'$, then $R(w, j)$ can be computed efficiently using the recurrence

$$R(w,j) = \max_{u \in I(w)} \{M(u,j) - \Delta(1), R(u,j) - t\}. \tag{1}$$

Value of t in the $Q(w, j)$ and $R(w, j)$ expressions mentioned is the gap extension cost. Matrices M and Q are initialized as described earlier and $R(\theta, j) = -\infty$ for $j \in [1, m]$.

While filling any column j of matrix R, the $R(w, j)$ entries for $w \in V'$ are filled first using its original definition and the remaining entries for vertices in $V - V'$ are filled using Equation 1.

3.3. Handling variable length vertex labels

In the previous section, we assumed that the vertex labels are elements from Σ. We extend our algorithm in a straightforward manner to handle vertex labels from Σ^+. Consider the directed graph $G_a = (V_a, E_a, \gamma)$ derived from $G = (V, E)$ as follows. For each vertex $v \in V$, with label $\gamma(v) = (b_1, \ldots, b_r)$, include a chain of r corresponding vertices $v_1, \ldots, v_r$ in G_a with directed edges from v_i to v_{i+1}. Label of v_i is given by $\gamma(v_i) = b_i$. Clearly $|V_a| = \sum_{v \in V} |\gamma(v)|$. For each directed edge $(u, v) \in E$, we include a directed edge $(u_{|\gamma(u)|}, v_1)$ to E_a. Hence $|E_a| = |E| + \sum_{v \in V} (|\gamma(v)| - 1)$. A linear ordering of V can be extended easily to obtain a linear ordering of V_a by replacing each vertex v in the linear ordering by the corresponding chain of vertices $v_1, \ldots, v_{|\gamma(v)|}$ in the new ordering. For any vertex $v \in V$, clearly the corresponding vertices $v_2, \ldots, v_{|\gamma(v)|} \in V_a$ are in order and v_1 is in order vertex if and only if v is in order vertex. That is, for G_a, the set of vertices not in order is given by $V'_a = \{v_1 | v \in V'\}$. Hence $|V'_a| = |V'|$. The DP matrix M_a is of size $|V_a| \times m$ and it is filled in the same manner.

4. COMPLEXITY ANALYSIS

In this section, we first present complexity analysis of our algorithm. We also present analytical and experimental evaluation of the DAGification overhead incurred by the existing POA-based techniques. We recall that existing POA-based techniques suffer from a blowup in the input graph size due to DAGification preprocessing and thereby incur increased computational cost. This preprocessing is completely avoided by our algorithm.

4.1. Single literal vertex label case

We first analyze the algorithm for the simple case where the graph vertex labels are single literals. In the next section, we analyze the general setting. We note that the time for updating one column of M is the sum total of all in degrees of vertices in V, which is $O(E)$. Time to update one column of Q is $O(V)$. Time for updating one column of R is the sum of the time taken for all $v \in V'$ and the time taken for all $v \in V - V'$.

Downloaded by Univ Of Western Ontario from www.liebertpub.com at 09/12/18. For personal use only.

The first component is $O(V'E)$ and the second component is the sum of in degrees, which is again $O(E)$. Hence the total time for filling R is $O(mE(V'+1))$. If $V'=\phi$, which is the case when G is a DAG, then the runtime matches (Lee et al., 2002). We note that, in general, there are graphs where the number of vertices in V' can be $\Omega(V)$ for any ordering. The runtime for such graphs is $O(mVE)$ that matches the runtime of our basic algorithm.

If F is a subset of V with minimum cardinality and whose removal makes G acyclic, which is called a minimum feedback vertex set (MFVS), then the time taken is $O(m(f+1)E)$, where $f=|F|$. This is because, we can use the vertex ordering given by the topological sorting of the subgraph induced by V minus the vertices in F and then place the vertices in F in the beginning of this ordering. For this ordering, $V'=F$. Depending on f, the runtime could be smaller than the $O(mVE)$ time taken by our basic algorithm presented in Section 3.1. Although MFVS problem is NP-complete, approximation algorithms, parameterized algorithms, and efficient exact algorithms for special graph classes are known (Ueno et al., 1988; Li and Liu, 1999; Dinur and Safra, 2005; Chen et al., 2008; Baharev et al., 2015). We remark that any ordering with small V' can lead to improved performance. We also remark that if G is a simple directed path, then the alignment problem reduces to the alignment of two sequences, of lengths $|V|$ and m, respectively, and the time taken in this case is $O(mE)=O(mV)=O(mn)$, where $|V|=n$.

4.2. Variable length vertex label case

We now consider the general setting where vertex labels can be variable length sequences. In this case, the time for filling M is $O(mE_a(V'+1))=O(m(n+E)(V'+1))$, where $n=\sum_{v\in V}|\gamma(v)|$ is the sum total of the sizes of all vertex labels in G. It follows that if G has bounded feedback vertex set, then the runtime is $O(m(n+E))$. Furthermore, for constant length vertex labels, the time is $O(mE(V'+1))$, which reduces to $O(mE)$ for G with bounded feedback vertex set. We remark that if G is just a single vertex with a sequence label of length n and containing no edges, the alignment problem reduces to the standard sequence-to-sequence alignment. In this case, V-ALIGN takes $O(mn)$ time.

We do not require the precomputation of all-pair shortest edge distances in G_a during preprocessing. Instead, we can compute vertex weighted all-pair shortest paths in G and use them to obtain $\delta(u_i, v_j)$ for any pair of vertices u_i, v_j in G_a. For this, each vertex v in G assigned a weight $w(v)$ equal to the length of its label. That is, $w(v)=|\gamma(v)|$. Weight of a path in G is the sum of the vertex weights in the corresponding vertex sequence. Now we compute $\delta_w(u, v)$ for u, v in G that is the minimum weight of any path from u to v. We define $\delta_w(u, u)=0$ and $\delta_w(u, v)=+\infty$ if there is no path from u to v. For $u\neq v$, the shortest edge distance from u_i to v_j in G_a is now given by $\delta(u_i, v_j)=j+\delta_w(u, v)-i-|\gamma(v)|$. Clearly $\delta(u_i, u_j)=j-i$ when $j\geq i$. If $j<i$ and if $I_u\neq\phi$ is the set of in neighbors of u in G, then $\delta(u_i, u_j)=|\gamma(u)|+j-i+\min_{v\in I_u}\delta_w(u, v)$. From the description of the algorithm, it is clear that we require $\delta(v, w)$ precomputation only for $w\in V'$. Hence, computing δ_w for all pairs (u, w) with $w\in V'$ can be done using the standard Dijkstra's algorithm in $O(V'E+V'V\log V)$ time and the results can be stored in $O(V'V)$ space.

4.3. Time complexity for DAGification preprocessing

We present a simple theoretical result on the complexity of DAGification preprocessing used by existing POA-based techniques. This preprocessing is avoided by our algorithm. We assume that the directed graph $G=(V, E, \gamma)$ is connected and that the vertex labels are elements of the alphabet Σ. We first define a k-DAG of a directed graph G for $k\geq 1$. We say that a DAG $G'=(V', E', \gamma')$ is a k-DAG of G when the following holds: G contains a label sequence y of length k or less if and only if G' contains y. For a graph $G=(V, E, \gamma)$, clearly $G'=(V, \phi, \gamma)$ is a 1-DAG of G. A k-DAG of G with $k\geq 2$ may contain more vertices and edges than G. This is because additional vertex copies with the same vertex label are included in G' every time the same vertex is encountered during loop unrolling.

POA-based sequence technique first computes a k-DAG G' of the input graph G and then computes an optimal gapped alignment of the input sequence to some label sequence contained in G'. If the vertex labels in G are just the elements of the alphabet Σ, then the gapped alignment of an m length sequence requires a k-DAG of G with $k\geq m$. Aligning the m length sequence on a k-DAG $G'=(V', E', \gamma')$ requires $O(m|E'|)$ time (Lee et al., 2002). Hence the size of the k-DAG affects the alignment performance.

In the following we present a simple complexity result on computing k-DAG of a directed graph $G=(V, E, \gamma)$ assuming that vertex labels in G are elements of Σ and the labels are distinct. For simplicity, we assume that $\Sigma=V$ and $\gamma(v)=v$.

Downloaded by Univ Of Western Ontario from www.liebertpub.com at 09/12/18. For personal use only.

Theorem 1. *For directed graph $G=(V, E, \gamma)$, computing a 2-DAG of G having minimum number of vertices is NP-complete.*

Proof. Clearly G is assumed to be not a DAG. Let $V_f=\{u_1, \ldots, u_f\}$ be a feedback vertex set of G. Consider the graph G' obtained by adding f new vertices $\{v_1, \ldots, v_f\}$ to G where the label of v_i is $\gamma(u_i)$. Each directed edge (w, u_i) for any $u_i \in V_f$ is replaced in G' by a new edge (w, v_i). It is straightforward to verify that G' is a 2-DAG of G with $|V|+f$ vertices. Conversely, if $G''=(V'', E'', \gamma'')$ is a 2-DAG of G, then the set of vertex labels each of which is assigned to more than one vertex in G'' forms a feedback vertex set of G. The cardinality of this feedback vertex set is at most $|V''|-|V|$. The result now follows from the NP-completeness of MFVS computation on directed graphs. ■

Generating a k-DAG for an input graph G can nevertheless be done with a simple k-depth dfs traversal that explores k length paths in G (Novak et al., 2017). The output DAG, however, need not have the minimum number of vertices or edges. Each strongly connected component of G can be DAGified separately. For now, we assume G is strongly connected. A separate k-depth dfs starting from each vertex of G is performed. Each vertex of the DAG has an associated level. During the traversal through a vertex, its neighboring vertices in G and the connecting edges are added to the same level in the DAG unless the neighbor is already present at that level. Otherwise, an additional copy of the neighboring vertex is added to the next level and the connecting edge goes across the two levels. The traversal now proceeds from the newly added copy of the neighbor. Vertices are added to level 0 when encountered for the first time. Repeated invocations of an r-depth dfs from a vertex copy for the same r value are avoided by book keeping.

Downloaded by Univ Of Western Ontario from www.liebertpub.com at 09/12/18. For personal use only.

5. EXPERIMENTS

5.1. Size blowup of DAGification output graph for POA-based technique.

The DAGification preprocessing in POA-based technique blows up the number of vertices and edges in the DAGified graph (Novak et al., 2017). The size of the DAGified graph depends on the topology of the input graph G and the input sequence length m. If r is the length of the shortest directed cycle in G, called the girth of G, then the DAGification will unroll this cycle $\Theta(m/r)$ times. If the vertices in graph G have variable length sequence labels, then the cycle length is defined as the sum of the length of the vertex labels along the cycle. There are graphs where cycle unrolling can result in a blowup of vertices and edges by a multiplicative factor $\Theta(m/r)$. Such graphs are discussed later on in this section. The time complexity for POA-based technique in such cases is $O((m/r+1)mE)$. In the worst case, this can be $O(m^2E)$, which grows quadratically with the input sequence length. This time complexity is excluding the time required for the DAGification.

We conduct experiments to measure the actual blowup in the graph size due to DAGification for different input graphs. In Section 4.3, we showed that computing DAGified graph with minimum size is hard. Generating a k-DAG can nevertheless be done with a simple k-depth dfs traversal without guaranteeing minimum number of vertices or edges (Novak et al., 2017). This approach is also outlined in Section 4.3 and we use this approach here for DAGification.

The candidate set of graphs $\mathcal{C}$ used in our experiments consists of two classes of synthetic graphs. The first class consists of complete graphs K_n with n ranging from 1 to 5. A K_3 is shown in Figure 1D. Girth of any K_n is 1.

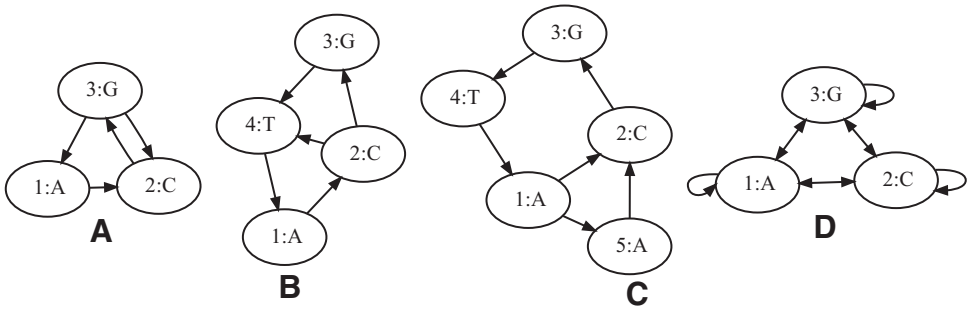

FIG. 1. Graphs with varying girth. Figure shows graphs with girth values ranging from 1 to 4.

The second class of graphs has larger girth values. Figure 1A–C shows the graphs that are present in the second class. Their girth values r range from 2 to 4. For each of these graphs, four copies are connected to form a cyclic chain with the same girth value. Clearly, for these graphs, the girth of the resulting chain graph is same as the girth of any of its constituent copies. These chain graphs are also included in the second class. Figure 2 shows a chain graph obtained from the girth 2 graph. In this graph, vertices 4, 7, and 10 are copies of vertex 1.

Figures 3–5 show the number of vertices and edges in the k-DAGification output graph for the input graphs in $\mathcal{C}$. The value of k depends on the length of the sequence to be aligned and k should be greater than the sequence length to allow for gaps in the alignment. In our experiments, we consider different k values from the range [10, 50] and from the range [100, 1000].

As shown in the plots in Figures 3–5., both edge set and vertex set cardinalities increase linearly with k after DAGification. The increase is lesser for graphs with larger girth as longer cycles are unrolled less number of times than shorter cycles for a given k. For example, Figure 5A and B shows that even though the chain of girth 4 and girth 2 graphs has 28 and 20 edges, respectively, the DAGified output for the latter has significantly higher size than the former for every k value.

From these plots, we see that the DAGification produces graphs with significantly larger size than the original input graph. This affects both the preprocessing cost and the subsequent alignment cost for POA-based technique. Recall that the worst case performance of V-ALIGN is $O(kVE)$ where k is the sequence length and V and E are the vertex and edge counts of the input graph. In contrast, the POA-based technique cost is $O(kE')$ where E' is the edge count of the DAGified output. From the mentioned plots, we see that E' is significantly more than $V \times E$ in most cases. For example, the chain of girth 2 graphs has $V \times E = 240$, where $E' = 11,117$ (DAG size in Figure 5B for $k = 1000$). Similarly, $V \times E = 12$ for girth 2 graph and $E' = 2222$ (DAG size in Figure 4B for $k = 1000$). For the complete graph K_5, $V \times E = 125$ and $E' = 24,975$ (DAG size for $k = 1000$). That is, already for sequence length $k = 1000$, the computational steps for POA-based techniques increase by roughly 100-fold or more as compared with V-ALIGN in several of these graphs. Figure 6 shows, for different values of k, the $E'/(VE)$ value averaged over all graphs in $\mathcal{C}$.

5.2. 1000 Genomes variation graphs

We conducted alignment experiments using V-ALIGN on the genome variation graphs constructed from the 1000 Genomes data. For this, we considered the GRCh37 reference genome (GRCh37, 2012) and the 1000 Genomes Variant Call Format data VCF:1000G phase-3 (1000Genome, 2013), which contains variations present in about 2500 genome samples. The *VG* tool (Variation Graph, 2017) was used to construct a genome variation graph that combined the reference genome and the VCF data. A separate graph was

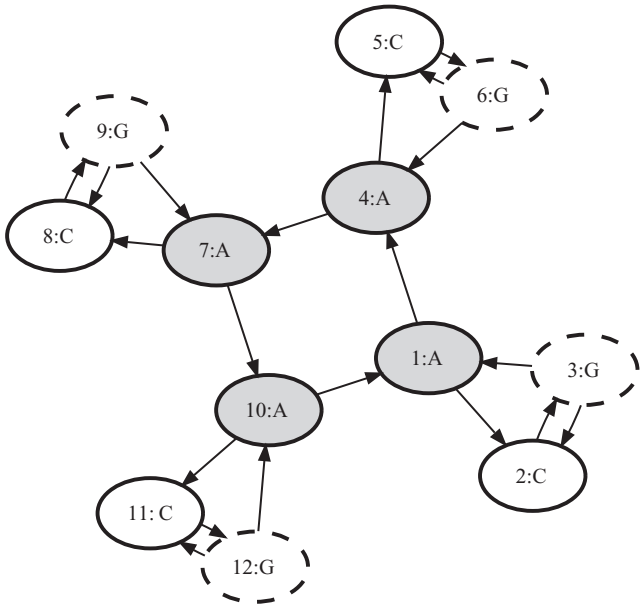

FIG. 2. A chain of girth = 2 graph (Fig. 1A).

Downloaded by Univ Of Western Ontario from www.liebertpub.com at 09/12/18. For personal use only.

Downloaded by Univ Of Western Ontario from www.liebertpub.com at 09/12/18. For personal use only.

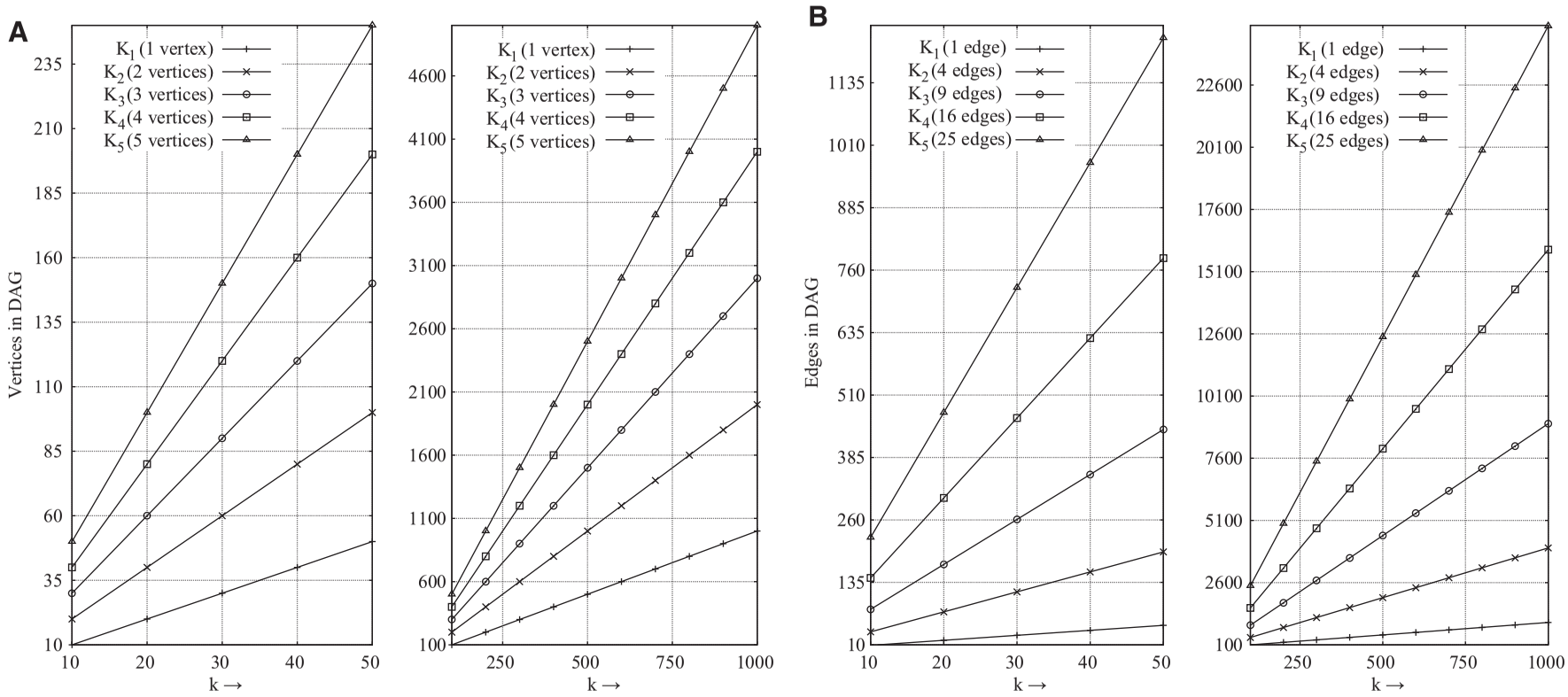

FIG. 3. DAGification overhead for POA-based techniques. Plot of the number of vertices **(A)** and edges **(B)** in the DAGification output for graphs K_1, K_2, K_3, K_4, and K_5 from $\mathcal{C}$ and for k values from the range [10, 50] and [100, 1000]. DAGs, directed acyclic graphs; POA, partial order alignment.

Downloaded by Univ Of Western Ontario from www.liebertpub.com at 09/12/18. For personal use only.

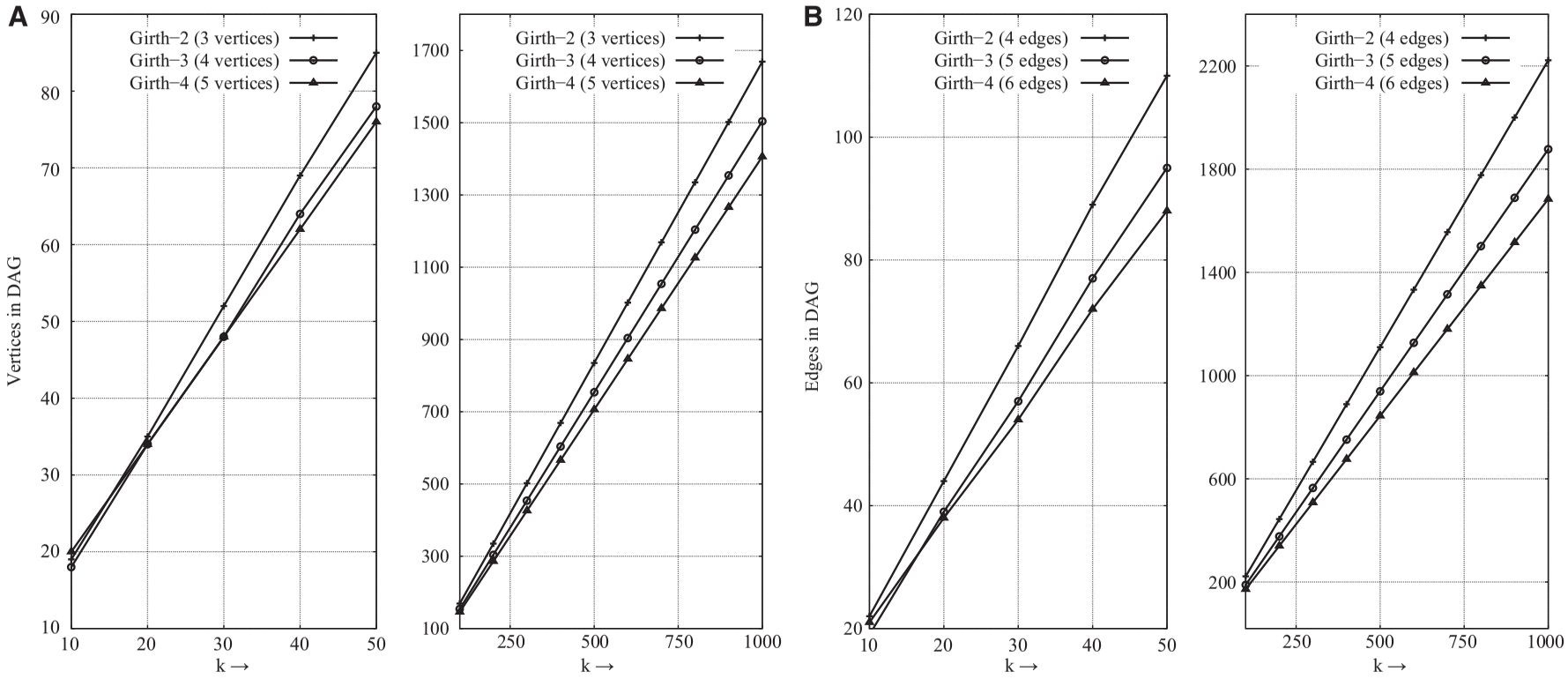

FIG. 4. DAGification overhead for POA-based technique. Plot of the number of vertices **(A)** and edges **(B)** in the DAGification output for the girth-r graphs from $\mathcal{C}$ and for k values from the range [10, 50] and [100, 1000].

Downloaded by Univ Of Western Ontario from www.liebertpub.com at 09/12/18. For personal use only.

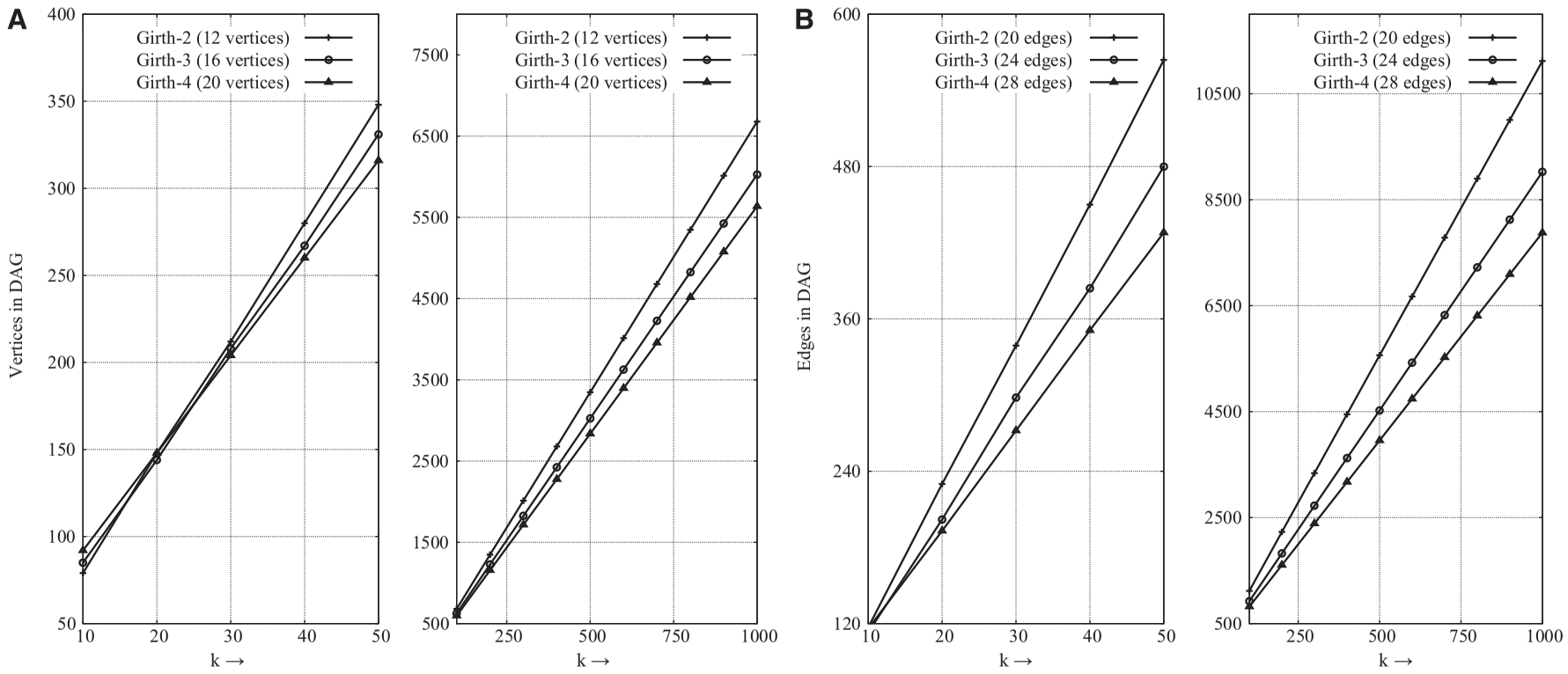

FIG. 5. DAGification overhead for POA-based technique. Plot of the number of vertices **(A)** and edges **(B)** in the DAGification output for the chain of girth-r graphs from $\mathcal{C}$ and for k values from the range [10, 50] and [100, 1000].

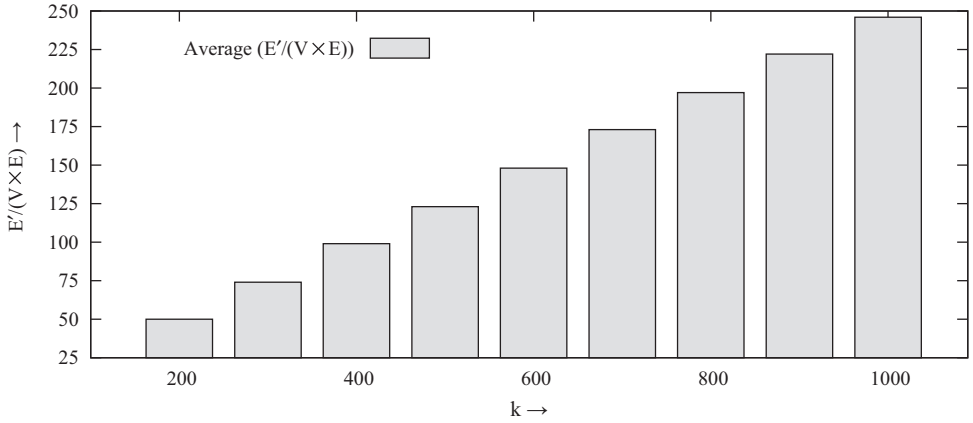

FIG. 6. Average $E'/(VE)$ value for graphs in $\mathcal{C}$ after k-DAGification for different k values.

Downloaded by Univ Of Western Ontario from www.liebertpub.com at 09/12/18. For personal use only.

constructed for each chromosome. The VG graph construction tool is part of the ongoing VG project and the current tool implementation constructs only acyclic graphs. To introduce cycles in these graphs, we additionally considered the short tandem repeat variant data (STR VCF) from the 1000 Genomes project (STR, 2017). The STR VCF is not part of the VCF:1000G phase-3 data but is available separately from the 1000 Genomes project site (FTP, 2017). The STR VCF contains many STRs where the RUs have highly varying repeat counts across genomes. For instance, Chromosome 11 contained RUs whose RPA values varied from 1 to 27. Such variants are naturally represented as cycles in the genome variation graph. We considered variants whose RPA values formed a contiguous range. Among them, we selected 20 RUs that ranked high on their RPA variance, where the variance for an RU is computed from all its RPA values present in the VCF. Details of these RUs are provided in Section 2 of the Supplementary Material. We then modified the chromosome graphs to incorporate the STR variations corresponding to these 20 RUs. This was done by introducing cycles on the coordinates corresponding to these 20 RUs in their respective graphs. After this, for each of these 20 RUs, we extracted a subgraph consisting of $\sim 10^3$ vertices from its respective graph that captured the neighborhood of the RU. These 20 subgraphs were used as the candidate graphs for the subsequent alignment experiments. Statistics of these 20 subgraphs are given in Section 3 of the Supplementary Material. Around 250 variations were captured in each of these graphs on average. In each case, the window of the reference sequence where the variations are located contained roughly 7.6×10^3 to 21×10^3 nucleotides.

The candidate sequences for alignment to these graphs, where each sequence is of length roughly 10^3 nucleotides, were constructed as follows. For each of the 20 RUs, a 10^3 nucleotide length window around the RU in the reference sequence GRCh37 was considered. For each such window, the variants from the 1000 Genomes VCF:1000G phase-3 data that are present inside it were also considered. Each reference window together with its variant data was input to the GATK alternative sequence generation tool (GATK, 2017) to create an alternative sequence. More details on the GATK tool usage is given in Section 4 of the Supplementary Material. As a result, 20 alternative sequences were generated, one for each RU, each with length about 10^3 nucleotides. Additional alternative sequences were then generated from these seed sequences by altering the RPA count of the corresponding RUs. For each of the 20 seed sequences, 9 additional sequences were thus generated by allowing 9 different repeat values for the corresponding RU. The 1000 Genomes STR VCF data were used to choose the modified repeat counts for the RUs.

In summary, our experiment consisted of 20 candidate graphs, and for each graph, a separate set of 10 sequences each of length roughly 10^3 is input for alignment. Each graph represented about 250 variations across a pool of roughly 2500 genome sequences having length in the range 7.6×10^3 to 21×10^3 (details are in Section 3 of the Supplementary Material). In terms of graph size, each graph contained roughly 10^3 vertices, 1.2×10^3 edges, and the total length of vertex subsequences in a graph is in the range 7.6×10^3 to 21×10^3. Since a graph has around 1.2×10^3 edges, it contains several potential paths from which an optimal alignment path needs to be computed by the alignment algorithm.

Aligning the input sequences to their corresponding candidate graphs using V-ALIGN produced exact alignments as desired. For this, V-ALIGN took roughly 10 milliseconds for the one-time preprocessing and

$\sim$1.38 seconds on average for the alignment on a standard desktop machine. If DAGification-based alignment approaches were used in place of V-ALIGN, intermediate DAGified graphs with significantly increased size would have to be constructed. Each candidate graph contains a cycle involving an RU. Since a candidate sequence is roughly 10^3 nucleotide long, the cycle involving the RU has to be unrolled accordingly. Since RU has two nucleotides in each graph, the DAGification would include at least 500 copies of the RU vertex. The DAGification-based alignment tool VG (VG, 2017) allows only input sequences of length at most 100 for alignment.

In the traditional approach of using a linear sequence reference as opposed to a graph reference, these experiments would result in alignments with significant gaps, and finding an exact alignment involving the right set of variants from a separate VCF data would be more complex. Such complexities are completely avoided in graph-based references. These graphs do not suffer from the artificial separation of reference and variants. Moreover, cycles in these graphs allow compact representation of many variations.

6. DISCUSSION

We provide a novel alignment algorithm V-ALIGN that aligns an input sequence to a genome variation graph. Existing POA-based techniques first perform a DAGification preprocessing that converts the genome variation graph into a DAG with increased size and the alignment is then performed on this DAG. DAGification complexity and the size of the resultant DAG also increase with the input sequence length. Increased size of the DAG leads to increased computational steps for the alignment. V-ALIGN avoids such expensive DAGification preprocessing and its overhead. V-ALIGN is based on a novel DP formulation that allows gapped alignment (constant, linear, or affine) directly on the input graph. The time to fill the DP table has linear dependence on the sizes of the sequence, the graph, and its feedback vertex set. Our experiments show that V-ALIGN achieves considerable saving in the computational steps compared with POA-based technique. The saving is even more significant for larger input sequence lengths.

V-ALIGN performs a one-time computation of (1) a feedback vertex set of the graph and (2) edge distances from the feedback vertices to other vertices in the graph. This computation is independent of the input sequence or its length. This is in contrast to the DAGification preprocessing by POA-based techniques that depends on the input sequence length. Genome variation graphs usually contain feedback vertex set with very small cardinality in comparison with the size of the graph. Nevertheless, improved algorithms and implementations for feedback vertex set computation can improve the overall performance. The current implementation of V-ALIGN does not focus on performance optimization using parallelization and hardware-dependent optimization techniques. This is left as future work.

Usual approaches for fast alignment of a short input sequence to a large target sequence follow efficient filtering of regions (subsequences) in the target sequence having high ''similarity'' with the short sequence and restricting expensive gapped alignment only to these regions. In the same manner, for aligning to a large graph G, alignment can be restricted to regions (subgraphs) of G having high ''similarity'' (Novak et al., 2017). Such subgraphs can have considerably lesser number of vertices and edges than G and this can lead to faster alignment. The time complexity for V-ALIGN in this case depends on the size of the filtered subgraph. Here, we do not require recomputation of the shortest edge distances between vertex pairs in the subgraph. V-ALIGN can instead use the precomputed values from G. We note that the shortest distances with respect to G can only be better than the distances with respect to its subgraph. Since the target graph here is still G, improved shortest paths with respect to G lead to improved alignment of the sequence to G. With regard to the feedback vertex set used by V-ALIGN, we recall that V-ALIGN works with a linear ordering of the vertices based on the feedback vertex set. For the filtered subgraph, we could proceed with the ordering induced by the vertices in filtered subgraph. Alternatively, we could recompute an improved feedback vertex set for the subgraph and a corresponding vertex ordering. This is a matter of choice based on the overall performance considerations.

We performed alignment experiments on genome subgraphs constructed from the 1000 Genomes data. These graphs contained cycles to represent STRs whose RPA had a large variation in the population. Alignment of alternative sequences that were also generated from the 1000 Genomes data resulted in exact alignments on the graphs. This is because genome variation graphs provide a unified and compact representation of the reference as well as the variant information, and paths in the graph correspond to different variant combinations. Computing these alignments would be more complex in the traditional approach of

Downloaded by Univ Of Western Ontario from www.liebertpub.com at 09/12/18. For personal use only.

working with a separate linear reference and variant information. Focus of our work is a novel alignment algorithm on general graphs with cycles. These experiments on 1000 Genomes variation graphs that contain cycles to capture STRs were aimed at exhibiting the strength of V-ALIGN for aligning input sequences of any length to such graphs without requiring any DAGification preprocessing. It would be worthwhile to explore the possibility of incorporating other structural variations from the 1000 Genomes data in the graph with the help of cycles and perform similar alignment experiments on the resulting target graph using V-ALIGN.

Sequence-to-sequence alignment is one of the fundamental problems in genomics. Aligning an n length sequence to a target sequence of length m can be performed in $O(mn)$ time using the classical Smith–Waterman algorithm (Smith and Waterman, 1981; Gotoh, 1982). If the target is an acyclic graph (partial order graph) having m edges and with single literal vertex labels, the existing POA-based technique computes optimal alignment in $O(mn)$ time (Lee et al., 2002). Our work generalizes this fundamental alignment problem to general target graphs that can contain cycles. Our algorithm V-ALIGN computes an optimal alignment in $O(fmn)$ time where the additional factor f is the size of feedback vertex set of the graph. For target graph with bounded feedback vertex set, the runtime is $O(mn)$. Consequently, for acyclic graphs, the V-ALIGN runtime matches the existing $O(mn)$ time. If the target is an m length sequence, then it can be viewed as a simple path on m vertices and $m-1$ edges. In this case, our runtime thus matches the classical $O(mn)$ bound.

We showed that for graphs with bounded feedback vertex set, V-ALIGN runs in $O(m(n+E))$ time to align an m length sequence, where n is the total size of all the vertex labels in the graph. This bound matches the best known runtime for acyclic graphs. The number of cycles introduced to capture repetitive regions in the genome could be significantly smaller than the total genome size. Hence, in practice, the V-ALIGN runtime for general genome graphs could compare well with the POA-based techniques for the acyclic graphs.

ACKNOWLEDGMENT

The authors would like to acknowledge the contribution of Shreyansh Chhajer to the V-ALIGN tool implementation.

AUTHOR DISCLOSURE STATEMENT

The authors declare that no competing financial interests exist.

REFERENCES

1000Genome. 2013. 1000 genome vcf. Available at: ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502. Accessed April 15, 2017.

Baharev, A., Schichl, H., Neumaier, A., et al. 2015. An exact method for the minimum feedback arc set problem. *Univ. Vienna* 10, 35–60.

Chen, J., Liu, Y., Lu, S., et al. 2008. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM* 55, 21.

Compeau, P.E., Pevzner, P.A., and Tesler, G. 2011. How to apply de bruijn graphs to genome assembly. *Nat. Biotechnol.* 29, 987–991.

De Bruijn, N.G. 1946. A combinatorial problem. KNAW 49, 758–764.

Dilthey, A., Cox, C., Iqbal, Z., et al. 2015. Improved genome inference in the MHC using a population reference graph. *Nat. Genet.* 47, 682–688.

Dinur, I., and Safra, S. 2005. On the hardness of approximating minimum vertex cover. *Ann. Math.* 162, 439–485.

FTP. 2017. The international genome sample resource. Available at: ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/supporting/strs/. Accessed April 15, 2017.

GATK. 2017. Genome analysis toolkit. Available at: https://software.broadinstitute.org/gatk. Accessed April 15, 2017.

Gotoh, O. 1982. An improved algorithm for matching biological sequences. *J. Mol. Biol.* 162, 705–708.

Graphviz. 2017. Graph Visualization Software. Available at: http://www.internationalgenome.org/announcements/short-tandem-repeats-added-1000-genomes-release-ashg14-2014-10-18. Accessed April 15, 2017.

Downloaded by Univ Of Western Ontario from www.liebertpub.com at 09/12/18. For personal use only.

GRCh37. 2012. Reference genome. Available at: ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/phase2_reference_assembly_sequence. Accessed April 15, 2017.

Lander, E.S., Linton, L.M., Birren, B., et al. 2001. Initial sequencing and analysis of the human genome. *Nature* 409, 860–921.

Lee, C., Grasso, C., and Sharlow, M.F. 2002. Multiple sequence alignment using partial order graphs. *Bioinformatics* 18, 452–464.

Li, D., and Liu, Y. 1999. A polynomial algorithm for finding the minimum feedback vertex set of a 3-regular simple graph. *Acta Math. Sci.* 19, 375–381.

Myers, E.W. 2005. The fragment assembly string graph. *Bioinformatics* 21(suppl 2), ii79–ii85.

Novak, A.M., Hickey, G., Garrison, E., et al. 2017. Genome graphs. Available at: www.biorxiv.org. Accessed April 15, 2018.

Paten, B., Diekhans, M., Earl, D., et al. 2011a. Cactus graphs for genome comparisons. *J. Comput. Biol.* 18, 469–481.

Paten, B., Earl, D., Nguyen, N., et al. 2011b. Cactus: Algorithms for genome multiple sequence alignment. *Genome Res.* 21, 1512–1528.

Paten, B., Herrero, J., Beal, K., et al. 2008. Enredo and pecan: Genome-wide mammalian consistency-based multiple alignment with paralogs. *Genome Res.* 18, 1814–1828.

Pevzner, P.A., Tang, H., and Tesler, G. 2004. De novo repeat classification and fragment assembly. *Genome Res.* 14, 1786–1796.

Smith, T.F., and Waterman, M.S. 1981. Identification of common molecular subsequences. *J. Mol. Biol.* 147, 195–197.

STR. 2017. The international genome sample resource. Available at: http://www.internationalgenome.org/announcements/short-tandem-repeats-added-1000-genomes-release-ashg14-2014-10-18. Accessed April 15, 2017.

The Computational Pan-Genomics Consortium. 2018. Computational pan-genomics: Status, promises and challenges. *Brief. Bioinform* 19, 118–135.

Ueno, S., Kajitani, Y., and Gotoh, S. 1988. On the nonseparating independent set problem and feedback set problem for graphs with no vertex degree exceeding three. *Discrete Math.* 72, 355–360.

V-ALIGN. 2017. V-ALIGN Software. Available at: https://github.com/tcsatc/valign. Accessed April 10, 2018.

Variation Graph. 2017. VG Documentation. Available at: https://github.com/vgteam/vg/wiki/working-with-a-whole-genome-variation-graph. Accessed April 15, 2017.

VG. 2017. Whole genome variation graph. Available at: https://github.com/vgteam/vg/wiki/working-with-a-whole-genome-variation-graph. Accessed April 15, 2017.

Address correspondence to:
Dr. Naveen Sivadasan
TCS Research
Hyderabad 500081
India

E-mail: naveen.sivadasan@tcs.com

Downloaded by Univ Of Western Ontario from www.liebertpub.com at 09/12/18. For personal use only.