JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 27, Number 3, 2020
© Mary Ann Liebert, Inc.
Pp. 317–329
DOI: 10.1089/cmb.2019.0511

Graph Traversal Edit Distance and Extensions

ALI EBRAHIMPOUR BOROOJENY,[1] AKASH SHRESTHA,[1] ALI SHARIFI-ZARCHI,[2]
SUZANNE RENICK GALLAGHER,[1] S. CENK SAHINALP,[3] and HAMIDREZA CHITSAZ[1]

ABSTRACT

Many problems in applied machine learning deal with graphs (also called networks), including social networks, security, web data mining, protein function prediction, and genome informatics. The kernel paradigm beautifully decouples the learning algorithm from the underlying geometric space, which renders graph kernels important for the aforementioned applications. In this article, we give a new graph kernel, which we call graph traversal edit distance (GTED). We introduce the GTED problem and give the first polynomial time algorithm for it. Informally, the GTED is the minimum edit distance between two strings formed by the edge labels of respective Eulerian traversals of the two graphs. Also, GTED is motivated by and provides the first mathematical formalism for sequence co-assembly and de novo variation detection in bioinformatics. We demonstrate that GTED admits a polynomial time algorithm using a linear program in the graph product space that is guaranteed to yield an integer solution. To the best of our knowledge, this is the first approach to this problem. We also give a linear programming relaxation algorithm for a lower bound on GTED. We use GTED as a graph kernel and evaluate it by computing the accuracy of a support vector machine (SVM) classifier on a few data sets in the literature. Our results suggest that our kernel outperforms many of the common graph kernels in the tested data sets. As a second set of experiments, we successfully cluster viral genomes using GTED on their assembly graphs obtained from de novo assembly of next-generation sequencing reads.

Keywords: assembly graph, clustering genera, co-assembly, de novo variation detection, graph comparison, graph kernel.

1. INTRODUCTION

Networks, or graphs as they are called in mathematics, have become a common tool in modern biology. Biological information from DNA sequences to protein interaction to metabolic data to the shapes of important biological chemicals is often encoded in networks.

One goal in studying these networks is to compare them. We might want to know whether two DNA assembly graphs produce the same final sequences or how close the protein interaction networks of two related species are. Such comparisons are difficult owing to the fact that determining whether two graphs have an identical structure with different labels or vertex ordering is a problem in NP-complete complexity

[1]Department of Computer Science, Colorado State University, Fort Collins, Colorado.
[2]Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.
[3]National Cancer Institute, NIH, Bethesda, Maryland.

class, for which no polynomial algorithm is known. Therefore, any comparisons will need to focus on specific aspects of the graph.

Here, we present the notion of *graph traversal edit distance (GTED)* as presented by the authors in Boroojeny et al. (2018), a new method of comparing two networks. Informally, GTED gives a measure of similarity between two directed Eulerian graphs with labeled edges by looking at the smallest edit distance that can be obtained between strings from each graph via an Eulerian traversal. GTED was inspired by the problem of *differential genome assembly*, determining if two DNA assembly graphs will assemble to the same string. In the differential genome assembly problem, we have the de Bruijn graph representations of two (highly) related genome sequence data sets, where each edge e represents a substring of size k from *reads* extracted from these genome sequences (e.g., one from a cancer tissue and the other from the normal tissue of the same individual), and its multiplicity represents the number of times its associated substring is observed in the reads of the respective genome sequence. In this formulation, each vertex represents the $k-1$ length prefix of the label of its outgoing edges and the $k-1$ length suffix of the label of its incoming edges. Thus, the labels of all incoming edges of a vertex (respectively, all outgoing edges) are identical with the exception of their first (last) symbol.

Differential genome assembly has been introduced to bioinformatics in two flavors: (1) *reference genome free* version (Li et al., 2011; Movahedi et al., 2012; Iqbal et al., 2012; Taghavi et al., 2013; Movahedi et al., 2016) and (2) *reference genome dependent* version, which, in its most general form, is NP-hard according to Hormozdiari et al. (2011). Both versions of the problem are attracting significant attention in biomedical applications (e.g., Mak, 2011; Jones, 2012) due to the reduced cost of genome sequencing (now approaching $1000 per genome sample) and the increasing needs of cancer genomics where tumor genome sequences may significantly differ from the normal genome sequence from the same individual through single symbol edits (insertions, deletions, and substitutions) as well as block edits (duplications, deletions, translocations, and reversals).

In addition to comparing assembly graphs, GTED can also be used to compare other types of networks. GTED yields a (pseudo-)metric for general graphs because it is based on the edit distance metric. Hence, it can be used as a graph kernel for a number of classification problems. GTED is the first mathematical formalism in which global traversals play a direct role in the graph metric. In this article, we give a polynomial time algorithm using linear programming (LP) that is guaranteed to yield an integer solution. We use that as a graph kernel and evaluate the performance of our new kernel in support vector machine (SVM) classification over a few data sets. We also use GTED for clustering of viral genomes obtained from de novo assembly of next-generation sequencing reads. Note that GTED is a global alignment scheme that is not immediately scalable to full-size large genomes, like all other global alignment schemes such as Needleman–Wunsch. However, GTED can form the mathematical basis for scalable heuristic comparison of full-size large genomes in the future.

1.1. Related work

Many problems in applied machine learning deal with graphs, ranging from web data mining (Inokuchi et al., 2003) to protein function prediction (Borgwardt et al., 2005). Some important application domains are biological networks such as regulatory networks, sequence assembly and variation detection, and structural biology and chemoinformatics where graphs capture structural information of macromolecules. For instance, machine learning algorithms are often used to screen candidate drug compounds for safety and efficacy against specific diseases and also for repurposing of existing drugs (e.g., Kubinyi, 2003). Kernel methods elegantly decouple data representation from the learning part; hence, graph learning problems have been studied in the kernel paradigm (Gärtner, 2002). Following the work by Gärtner (2002), other graph kernels have been proposed in the literature (Vishwanathan et al., 2010).

A graph kernel $k(G_1, G_2)$ is a (pseudo-)metric in the space of graphs. A kernel captures a notion of similarity between G_1 and G_2. For instance for social networks, k may capture similarity between their clustering structures, degree distribution, and so on. For molecules, similarity between their sequential/functional domains and their relative arrangements is important. A kernel is usually computed from the adjacency matrices of the two graphs, but it must be invariant to the ordering (permutation) of the vertices. That property has been central in the graph kernels literature.

Existing graph kernels that are vertex permutation invariant use either local invariants, such as counting the number of triangles and squares that appear in G as subgraphs, or spectral invariants captures as functions of the eigenvalues of the adjacency matrix or the graph Laplacian. Essentially, different graph kernels ranging from random walks (Gärtner, 2002) to shortest paths (Borgwardt and Kriegel, 2005; Feragen et al., 2013) to Fourier transforms on the symmetric group (Kondor and Borgwardt, 2008) to multiscale Laplacian (Kondor and Pan, 2016) compute local, spectral, or multiscale distances. While most subgraph counting kernels are local (Shervashidze et al., 2009), most random walk kernels are spectral (Vishwanathan et al., 2010). Multiscale Laplacian (Kondor and Pan, 2016), Weisfeiler–Lehman kernel (Shervashidze et al., 2011), and propagation kernel (Neumann et al., 2016) are among the multiscale kernels.

In this article, we introduce a graph kernel based on comparison of global Eulerian traversals of the two graphs. To the best of our knowledge, our formalism is the first to capture global architectures of the two graphs as well as their local structures. Our kernel is based on the GTED introduced in this article. We show that a lower bound for GTED can be computed in polynomial time using the LP relaxation of the problem. In practice, the linear program often yields an integer solution, in which case the computed lower bound is actually equal to GTED. In Ebrahimpour Boroojeny et al. (2020), we have explained in detail the instructions for working with the software developed for computing GTED. The software is freely available under open source (https://github.com/Ali-E/GTED).

2. PROBLEM DEFINITION

Due to diversity of applications, input graphs can be obtained as molecular structure graphs, social network graphs, systems biology networks, or sequence assembly graphs such as de Bruijn graphs (Pevzner et al., 2001), A-Bruijn graphs (Pevzner et al., 2004), positional de Bruijn graphs (Ronen et al., 2012), string graphs (Myers, 1995), or implicit string graphs (Simpson and Durbin, 2010) among numerous alternatives. Our GTED is inspired by those applications and can potentially be adapted to any of those frameworks. However, we choose below a general, convenient representative definition for the problem. For the sake of brevity, we assume throughout this article that the input graph has one strongly connected component.

Definition 1 (Edge-labeled Eulerian Graph). *Let Σ be a finite alphabet. We call a tuple $A=(V, E, M, L, \Sigma)$ an edge-labeled Eulerian graph, in which*

- *$G=(V, E)$ is a strongly connected directed graph,*
- *$M : E \to \mathbb{N}$ specifies the edge multiplicities,*
- *$L : E \to \Sigma$ specifies the edge labels,*

iff G with the corresponding edge multiplicities, M, is Eulerian. That is, G contains a cycle (or path from a specified source to a sink) that traverses every edge $e \in E$ exactly $M(e)$ times. Throughout this article, we mean M-compliant Eulerian by an Eulerian cycle (path) in A.

Figure 1 demonstrates an example edge-labeled Eulerian graph for the circular sequence *ACAGACAT* in the alphabet $\Sigma=\{\text{A, C, G, T}\}$. The sequence of edge labels over the Eulerian cycle formed by c_1 followed by c_2 yields the original sequence. The following definition makes a connection between Eulerian cycles and different sequences they spell.

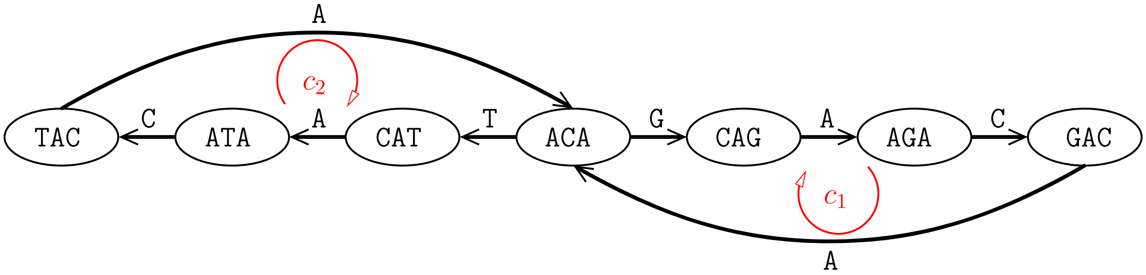

FIG. 1. Edge-labeled Eulerian graph. An edge-labeled Eulerian graph $A=(V, E, M, L, \{\text{A, C, G, T}\})$ obtained from the $k=4$ de Bruijn graph $G=(V, E)$ for the *circular* sequence *ACAGACAT* (Jones and Pevzner, 2004). Vertices, *V*, correspond to $(k-1)$-mers and edges correspond to *k*-mers. In this case, all the edges have multiplicity one, that is, $M \equiv 1$. Edge labels, *L*, show the *k*th nucleotide in the associated *k*-mers.

Definition 2 (Eulerian Language). *Let $A=(V, E, M, L, \Sigma)$ be an edge-labeled Eulerian graph. Define the word ω associated with an Eulerian cycle (path) $c=(e_0, \dots, e_n)$ in A to be the word*

$$\omega(c)=L(e_0)\dots L(e_n) \in \Sigma^*. \tag{1}$$

The language of A is then defined to be

$$\mathcal{L}(A)=\{\omega(c) \mid c \text{ is an Eulerian cycle (path) in } A\} \subset \Sigma^*. \tag{2}$$

We now define GTED.

Problem 1 (Graph Traversal Edit Distance). *Let A_1 and A_2 be two edge-labeled Eulerian graphs. We define the edit distance between A_1 and A_2 by*

$$d(A_1, A_2)= \min_{\substack{\omega_1 \in \mathcal{L}(A_1)\\ \omega_2 \in \mathcal{L}(A_2)}} d(\omega_1, \omega_2), \tag{3}$$

in which $d(\omega_1, \omega_2)$ is the Levenshtein edit distance between two strings ω_1 and ω_2. Throughout this article, edit operations are single alphabet symbol insertion, deletion, and substitution, and the Levenshtein edit distance is the minimum number of such operations to transform ω_1 to ω_2 (Levenshtein, 1966).

Note that $d(A_1, A_2)$ is the minimum of such edit distances over the words of possible Eulerian cycles (paths) in A_1 and A_2. Note that GTED is almost a metric but not a metric since there are A_1, A_2 such that $d(A_1, A_2)=0$ even though $A_1 \neq A_2$. For instance, let A_1 be an arbitrary Eulerian graph and A_2 be a cycle graph whose edge labels are the same as an arbitrary Eulerian cycle in A_1. As a result, the GTED is different from the graph edit distance because the latter is a metric, whereas the former is not.

3. METHODS

3.1. Brute force computation of GTED

It is clear that there are algorithms, although with exponential running time, that enumerate all Eulerian cycles in a graph. Through brute force Needleman–Wunsch alignment of the words of every pair of Eulerian cycles in A_1 and A_2, we can compute the edit distance right from the definition. De Bruijn, van Aardenne-Ehrenfest, Smith, and Tutte proved the de Bruijn–van Aardenne–Ehrenfest–Smith–Tutte (BEST) theorem (Tutte and Smith, 1941; van Aardenne-Ehrenfest and de Bruijn, 1987), which counts the number of different Eulerian cycles in A as

$$ec(A)=t_w(A)\prod_{v\in V}(\deg(v)-1)!, \tag{4}$$

in which $t_w(A)$ is the number of arborescences directed toward the root at a fixed vertex w, and deg is the indegree (or equally outdegree) considering multiplicities. The number of Eulerian cycles $ec(A)$ is exponentially large in general. Therefore, the naive brute force algorithm is intractable.

3.2. GTED as a constrained shortest path problem

The conventional string alignment problem can be transformed into a shortest path problem in an alignment graph, which is obtained by adding appropriate edges to the Cartesian product of the two string graphs. Figure 2 illustrates an example; further details can be found in a bioinformatics textbook such as the one by Jones and Pevzner (2004). Analogously, the GTED $d(A_1, A_2)$ can be written as the length of the shortest cycle (or path from a designated source to a designated sink) in the alignment graph defined below, whose projection onto A_1 and A_2 is Eulerian. To state that fact in Lemma 1, we need

Definition 3 (Alignment Graph). *Let $A_1=(V_1, E_1, M_1, L_1, \Sigma_1)$ and $A_2=(V_2, E_2, M_2, L_2, \Sigma_2)$ be two edge-labeled Eulerian graphs. Define the alignment graph between A_1 and A_2 to be $\mathcal{AG}(A_1, A_2)=(V_1\times V_2, E)$, in which E is a collection of horizontal, vertical, and diagonal edges as follows:*

- *Vertical:* $\forall\, e_1=(u_1, v_1) \in E_1$ *and* $u_2 \in V_2 :\ e_1\times u_2=[(u_1, u_2), (v_1, u_2)] \in E,$
- *Horizontal:* $\forall\, u_1 \in V_1$ *and* $e_2=(u_2, v_2) \in E_2 :\ u_1\times e_2=[(u_1, u_2), (u_1, v_2)] \in E,$
- *Diagonal:* $\forall\, e_1=(u_1, v_1) \in E_1$ *and* $e_2=(u_2, v_2) \in E_2 :\ [(u_1, u_2), (v_1, v_2)] \in E.$

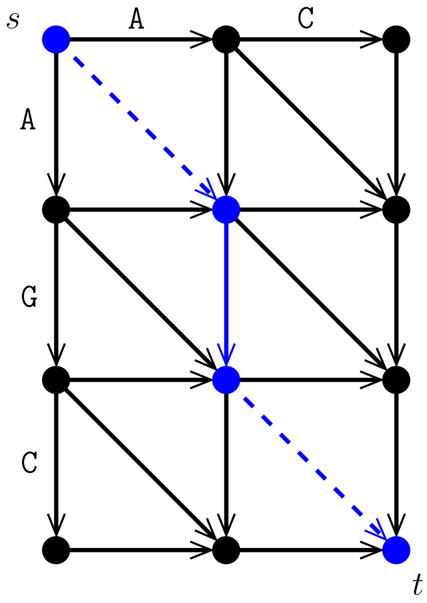

FIG. 2. Conventional alignment graph. The alignment graph for AC versus AGC. Those edges that correspond to matches are in dashed lines (cost of a match is often 0). Solid lines show substitutions and indels, which usually have a positive cost. The edit distance is the shortest distance from s to t in this graph (shown in blue).

There is a cost $\delta : E \to \mathbb{R}$ associated with each edge of $\mathcal{AG}$ based on edit operation costs. Horizontal and vertical edges correspond to insertion or deletion and diagonal edges correspond to match or mismatch (substitution). A diagonal edge $[(u_1, u_2), (v_1, v_2)]$ is a match iff $L(u_1, v_1) = L(u_2, v_2)$ and a mismatch otherwise.

The following Lemma states the fact that GTED is equivalent to a constrained shortest path problem in the alignment graph.

Lemma 1. *For any two edge-labeled Eulerian graphs $A_1 = (V_1, E_1, M_1, L_1, \Sigma_1)$ and $A_2 = (V_2, E_2, M_2, L_2, \Sigma_2)$,*

$$d(A_1, A_2) = \begin{array}{ll} \underset{c}{\textit{minimize}} & \delta(c) \\ \textit{subject to} & c \textit{ is a cycle (path) in } \mathcal{AG}(A_1, A_2), \\ & \pi_i(c) \textit{ is an Eulerian cycle (path) in } A_i \textit{ for } i = 1,\ 2, \end{array} \quad (5)$$

in which $\delta(c)$ is the total edge-cost (edit cost) of c, and π_i is the projection onto the ith *component graph.*

Proof. For every pair (c_1, c_2), in which c_i is an Eulerian cycle (path) in A_i, there are possibly multiple c's with $\pi_i(c) = c_i$, whose minimum total edge-cost is $d(\omega(c_1), \omega(c_2))$. Therefore, the result of the minimization in Equation (5) is not more than $d(A_1, A_2)$, that is, the right-hand side is less than or equal to $d(A_1, A_2)$. Conversely, every c that satisfies the constraints in Equation (5) gives rise to an Eulerian pair $(c_1, c_2) = (\pi_1(c), \pi_2(c))$ and $\delta(c) \geq d(\omega(c_1), \omega(c_2)) \geq d(A_1, A_2)$, that is, the right-hand side is greater than or equal to $d(A_1, A_2)$. ■

3.3. Lower bound via LP relaxation

Lemma 1 easily transforms our problem into an integer linear program (ILP) as the projection operator π_i is linear and imposing path connectivity/cycle is also linear. More precisely, consider two edge-labeled Eulerian graphs A_1 and A_2 with the alignment graph $\mathcal{AG}(A_1, A_2) = (V_1 \times V_2, E)$, and let ∂ be the boundary operator, $\partial(e) = v - u$ for an edge $e = (u, v)$, which is defined in detail below. Our algorithm consists in solving the LP relaxation of that ILP,

$$\begin{array}{ll} \underset{x\in\mathbb{R}^{|E|}}{\text{minimize}} & \sum_{e\in E} x_e\delta(e) \\ \text{subject to} & \sum_{e\in E} x_e\partial(e)=0 \quad (\text{or sink } - \text{ source}), \\ & \forall\, e\in E, \quad x_e\geq 0, \\ & \text{for}\, i=1,2, \forall\, f\in E_i, \quad \sum_{e\in E} x_e I_i(e,f)=M_i(f), \end{array} \tag{6}$$

in which indicator function $I_1(e,f)=1$ iff $e=f\times v_2$ or $e=[(u_1,u_2),(v_1,v_2)]$ with $f=(u_1,v_1)$; otherwise, $I_1(e,f)=0$. Similarly, $I_2(e,f)=1$ iff $e=v_1\times f$ or $e=[(u_1,u_2),(v_1,v_2)]$ with $f=(u_2,v_2)$; otherwise, $I_2(e,f)=0$. The linear program above is not guaranteed to give an integer solution; however, we have observed integer solutions in many scenarios. Nevertheless, the solution of Equation (6) is a lower bound for GTED.

3.4. Algorithm for GTED

The following theorem is the main result of this article, which bridges the gap between GTED and another LP formulation which we will show is guaranteed to have an exact integer solution. Hence, GTED has a polynomial time algorithm explained as a linear program; Corollary 1 states that fact below.

Theorem 1 (Graph Traversal Edit Distance). *Consider two edge-labeled Eulerian graphs $A_i=(V_i,E_i,M_i,L_i,\Sigma_i)$ with $G_i=(V_i,E_i)$ for $i=1,2$. Let T be the collection of two simplices in the triangulated $G_1\times G_2$ with one-faces in $\mathcal{AG}(A_1,A_2)$. In that case,*

$$d(A_1,A_2)= \begin{array}{ll} \underset{x\in\mathbb{R}^{|E|},\, y\in\mathbb{R}^{|T|}}{\textit{minimize}} & \sum_{e\in E} x_e\delta(e) \\ \textit{subject to} & x=x^{init}+[\partial]\,y, \\ & \forall\, e\in E, \quad x_e\geq 0, \end{array} \tag{7}$$

in which $[\partial]_{|E|\times|T|}$ is the matrix of the two-dimensional boundary operator in the corresponding homology and

$$x_e^{init}=\begin{cases} M_1(f) & \textit{if } e=f\times s_2 \\ M_2(f) & \textit{if } e=s_1\times f \\ 0 & \textit{otherwise} \end{cases}, \tag{8}$$

for arbitrary fixed $s_i\in V_i$ (source/sink in the case of path).

Proof. It is sufficient to show two things:

1. GTED is equal to the solution of the ILP version of the linear program in Equation (7).
2. The linear program in Equation (7) always yields an integer solution.

Using Lemma 1, we need to show that Equations (5) and (7) are equivalent for the first one. By construction, x^{init} corresponds to an Eulerian cycle (path) in A_1 followed by one in A_2, which specifies a cycle (path) in $\mathcal{AG}(A_1,A_2)$ whose projection onto A_i is Eulerian. It is sufficient to note that every cycle (path) whose projection onto A_i is Eulerian is homologous to x^{init}. To see that, let c be a cycle (path) whose projection onto A_i is Eulerian. First note that diagonal edges in c are homologous to the horizontal edge followed by the vertical edge in the corresponding cell. Hence, diagonal edges can be replaced by the horizontal followed by the vertical edge using the boundary operator $[\partial]$. Hence, without loss of generality, we assume c contains only horizontal and vertical edges.

If edges in c are $h_1,h_2,\ldots,h_m,k_1,k_2,\ldots,k_n$ such that $h_i=e_i\times s_2$ and $k_i=s_1\times f_i$ for $e_i\in E_1$ and $f_i\in E_2$, then we are done. We know that such c has exactly the same representation as x^{init}. If edges in c are $h_1,h_2,\ldots,h_m,k_1,k_2,\ldots,k_n$ such that $h_i=e_i\times v_2$ and $k_i=v_1\times f_i$ for $e_i\in E_1$ and $f_i\in E_2$ and possibly $v_1\neq s_1$ or $v_2\neq s_2$, then we can rotate the cycle through adding and subtracting a perpendicular translation edge and apply the boundary replacement operation to obtain a homologous cycle (path) of the form $h_1,h_2,\ldots,h_m,k_1,k_2,\ldots,k_n$ such that $h_i=e_i\times s_2$ and $k_i=s_1\times f_i$ for $e_i\in E_1$ and $f_i\in E_2$. Starting with an arbitrary c, we show how to obtain a homologous cycle (path) in the form of $h_1,h_2,\ldots,h_m,k_1,k_2,\ldots,k_n$ such that $h_i=e_i\times v_2$ and $k_i=v_1\times f_i$ for $e_i\in E_1$ and $f_i\in E_2$ through basic boundary replacement operations.

Essentially, we show that we can swap vertical and horizontal edges along c until we end up with all horizontal edges grouped right up front followed by all vertical edges grouped at the end. Suppose c contains k, h as a subpath for $h=e\times v_2$ and $k=u_1\times f$ and $e=(u_1, v_1)\in E_1$ and $f=(u_2, v_2)\in E_2$. The subpath k, h is homologous to h', k' in which $h'=e\times u_2$ and $k'=v_1\times f$ since the four edges $k, h, -k', -h'$ form the boundary of a square. Hence, we can replace k, h with h', k' in c to obtain a homologous cycle (path) c'. Performing a number of such vertical–horizontal swaps will yield the result. The second is going to be shown in the following sections. ■

Corollary 1 (GTED complexity). *The graph traversal edit distance is in P and can be solved in polynomial time from the linear program in Equation (7) that is guaranteed to give the integer solution.*

3.5. Total unimodularity

Using a recent result by Dey et al. (2011), we show that Equation (7) is guaranteed to yield an integer solution. The main reason is that the boundary operator matrix $[\partial]$ is totally unimodular, that is, all its square submatrices have a determinant in $\{0, \pm 1\}$. Therefore, all vertices of the constraint polytope in Equation (7) have integer coordinates; hence, the solution is integer.

Why is $[\partial]$ totally unimodular? According to Dey et al. (2011, Theorem 5.13), $[\partial]$ is totally unimodular iff the simplicial complex $G_1\times G_2$ has no Möbius subcomplex of dimension 2. For the sake of completeness, we include the definition of a Möbius complex below.

Definition 4 [Dey et al. (2011), Definition 5.9]. *A two-dimensional cycle complex is a sequence $\sigma_0\cdots\sigma_{k-1}$ of two simplices such that σ_i and σ_j have a common face iff $j=(i+1) \bmod k$ and that the common face is a one simplex. It is called a two-dimensional cylinder complex if orientable and a two-dimensional Mo̎bius complex if nonorientable.*

Lemma 2. *A triangulated graph product space $G_1\times G_2$ does not contain a Mo̎bius subcomplex, for directed graphs G_i with unidirectional edges.*

Proof. It is enough to observe that in $G_1\times G_2$, the orientation in one coordinate cannot flip. For brevity of presentation, we ignore triangulation for a moment and consider the rectangular cells. To the contrary, assume $G_1\times G_2$ contains a Möbius subcomplex $\sigma_0\cdots\sigma_{k-1}$ in which every σ_i is a rectangle $e_i\times f_i$, for $e_i\in E_1$ and $f_i\in E_2$. Since every σ_i and σ_{i+1} have a common edge and G_1, G_2 are directed graphs with unidirectional edges, either $e_{i+1}=e_i$ or $f_{i+1}=f_i$ but not both. In particular, $e_0=e_{k-1}$ or $f_0=f_{k-1}$. That is a contradiction because $\sigma_0\cdots\sigma_{k-1}$ is then a cylinder subcomplex (orientable) and not a Möbius subcomplex. ■

Lemma 2 together with [Dey et al. (2011), Theorem 5.13] assert that $[\partial]$ is totally unimodular. Therefore, Equation (7) always has an integer solution, hence the main result in Theorem 1.

Lack of Möbius subcomplexes in the product space of graphs, which are Möbius-free spaces, can also be seen from the fact that the homology groups of graph product spaces are torsion-free. The following section summarizes that characterization.

3.6. Homology theory of alignment graph

An alignment graph $\mathcal{AG}(A_1, A_2)$ is essentially a topological product space with additional triangulating diagonal edges corresponding to matches and mismatches. In other words, $\mathcal{AG}(A_1, A_2)$ can be regarded as a triangulation of the two-dimensional CW complex $G_1\times G_2$ (by horizontal, vertical, and diagonal edges). Note that $G_1\times G_2$ has zero-dimensional vertices (v_1, v_2), one-dimensional edges $e_1\times v_2$ and $v_1\times e_2$, and two-dimensional squares $e_1\times e_2$ for $v_i\in V_i$ and $e_i\in E_i$. We characterize below the homology groups of $G_1\times G_2$ using the Künneth's theorem. Note that G_i are obtained from edge-labeled graphs $A_i=(V_i, E_i, M_i, L_i, \Sigma_i)$.

Theorem 2 [Künneth; Vick (1994)]. *For graphs $G_i=(V_i, E_i)$, $i=1, 2$,*

$$H_m(G_1\times G_2, \mathbb{Z})\cong \bigoplus_{p+q=m} H_p(G_1, \mathbb{Z})\otimes H_q(G_2, \mathbb{Z}) \oplus \bigoplus_{r+s=m-1} Tor(H_r(G_1, \mathbb{Z}), H_s(G_2, \mathbb{Z})), \tag{9}$$

in which H_m is the mth *homology group and Tor is the torsion functor (Vick, 1994).*

Since $G_1 \times G_2$ is a two-dimensional CW complex, $H_m(G_1 \times G_2, \mathbb{Z}) \cong 0$ for $m > 2$. Clearly, $H_0(G_1 \times G_2, \mathbb{Z}) \cong \mathbb{Z}$ since $G_1 \times G_2$ is connected. According to the Künneth's theorem above and the fact that $\mathrm{Tor}(\mathbb{Z}, \mathbb{Z}) \cong 0$ and $\mathbb{Z}^k \otimes \mathbb{Z} \cong \mathbb{Z} \otimes \mathbb{Z}^k \cong \mathbb{Z}^k$ (Massey, 1991),

$$\begin{aligned} H_1(G_1 \times G_2) \cong\ & [H_1(G_1) \otimes H_0(G_2)] \oplus [H_0(G_1) \otimes H_1(G_2)] \oplus \mathrm{Tor}(H_0(G_1), H_0(G_2)) \\ \cong\ & [\mathbb{Z}^{n_1} \otimes \mathbb{Z}] \oplus [\mathbb{Z} \otimes \mathbb{Z}^{n_2}] \oplus \mathrm{Tor}(\mathbb{Z}, \mathbb{Z}) \cong \mathbb{Z}^{n_1} \oplus \mathbb{Z}^{n_2} \cong \mathbb{Z}^{n_1+n_2}, \end{aligned} \tag{10}$$

in which $n_i = 1 + |E_i| - |V_i|$. Note that $H_1(G_1, G_2)$ is torsion-free.

Using the Künneth's theorem above and the fact that the tensor product of groups $\otimes$ distributes over the direct sum $\oplus$, $H_2(G_i) \cong 0$, Tor of torsion-free groups is trivial, and $\mathbb{Z} \otimes \mathbb{Z} \cong \mathbb{Z}$, we obtain

$$\begin{aligned} H_2(G_1 \times G_2) \cong\ & [H_1(G_1) \otimes H_1(G_2)] \oplus \mathrm{Tor}(H_1(G_1), H_0(G_2)) \oplus \mathrm{Tor}(H_0(G_1), H_1(G_2)) \\ \cong\ & [\mathbb{Z}^{n_1} \otimes \mathbb{Z}^{n_2}] \oplus \mathrm{Tor}(\mathbb{Z}^{n_1}, \mathbb{Z}) \oplus \mathrm{Tor}(\mathbb{Z}, \mathbb{Z}^{n_2}) \cong \bigoplus_{i=1}^{n_1} \bigoplus_{j=1}^{n_2} \mathbb{Z} \otimes \mathbb{Z} \cong \mathbb{Z}^{n_1 n_2}. \end{aligned} \tag{11}$$

Note that $H_2(G_1, G_2)$ is torsion-free.

4. EXPERIMENTS

4.1. Using GTED to make a kernel

As mentioned earlier, since GTED is a measure of distance or dissimilarity between two graphs, we can use it to make a kernel of distance of pair of graphs in a data set, and this can be used for classification problems. We implemented a C++ program that generates the linear program for the problem. First, it builds the alignment graph $\mathcal{AG}$ for two given graphs $A_1 = (V_1, E_1)$ and $A_2 = (V_2, E_2)$, where V_i and E_i are vertices and edges of the ith graph. It begins with $|V_1| \times |V_2|$ vertices that are labeled as (v_1, v_2) for each $v_1 \in V_1$ and $v_2 \in V_2$. For each edge $(u_1, v_1) \in E_1$ and vertex $u_2 \in V_2$, we add the vertical edge $[(u_1, u_2), (v_1, u_2)]$ with a gap penalty δ_1 to our grid, $\mathcal{AG}$. We also add a horizontal edge $[(u_1, u_2), (u_1, v_2)]$ for each vertex $u_1 \in V_1$ and edge $(u_2, v_2) \in E_2$ with the same cost δ_1. Then, for each pair of edges $(u_1, v_1) \in E_1$ and $(u_2, v_2) \in E_2$, we add a diagonal edge $[(u_1, u_2), (v_1, v_2)]$, with a mismatch penalty δ_2 if (u_1, v_1) has a different label from (u_2, v_2), or a match bonus δ_3 if the labels are the same. The cost values are taken as arguments, with default values of $\delta_1 = \delta_2 = 1$ and $\delta_3 = 0$. This can be further extended to different penalties for insertion and deletion (i.e., different cost for horizontal and vertical edges).

The C++ program also creates a projection set for each edge in either of the input graphs. Each vertical edge $[(u_1, u_2), (v_1, u_2)]$ is added to the projection set of the edge $(u_1, v_1) \in E_1$, each horizontal edge $[(u_1, u_2), (u_1, v_2)]$ to the set of $(u_2, v_2) \in E_2$, and each diagonal edge $[(u_1, u_2), (v_1, v_2)]$ to projection sets of both $(u_1, v_1) \in E_1$ and $(u_2, v_2) \in E_2$.

Our program then extracts an LP problem from the alignment graph by assigning a variable x_i to the ith edge of $\mathcal{AG}$. The objective function minimizes weighted sum $\sum_{e \in E,\, \delta(e) > 0} x_e \delta(e)$. Then, the constraints will be generated. There are two different groups of constraints. The first group forces the vertices of the grid to have the same number of incoming edges and outgoing edges, forcing the output to be a cycle in the alignment graph. The second group forces the size of the projection set for each edge of the input graphs to be equal to its weight in that input graph, forcing the projection of the output to be Eulerian in both input graphs.

We used an academic license of Gurobi optimizer to solve the linear program. Since the variables are already supposed to be non-negative, it was not necessary to add inequalities to the LP for this purpose.

4.1.1. Data. We tested our graph kernel on four data sets. The Mutag data set consists of "aromatic and heteroaromatic nitro compounds tested for mutagenicity." Nodes in the graphs represent the names of the atoms. The Enzymes data set is a protein graph model of 600 enzymes from BRENDA database, which contains 100 proteins each from 6 Enzyme Commission top level classes (Oxidoreductases, Transferases, Hydrolases, Lyases, Isomerases, and Ligases). Protein structures are represented as nodes, and each node is connected to three closest proteins on the enzyme. The NCI1 data set is derived from PubChem website (pubchem.ncbi.nlm.nih.gov), which is related to screening of human tumor (nonsmall cell lung) cell line growth inhibition. Each chemical compound is represented by their corresponding molecular graph where nodes are various atoms (Carbon, Nitrogen, Oxygen, etc.) and edges are the bonds between atoms (single, double, etc.). The class labels on this data set is either active or inactive based on the cancer assay. The PTC data set is part of Predictive Toxicology Evaluation Challenge. This data set is composed of graphs representing chemical structure and their outcomes of biological tests for the carcinogenicity in male rats (MR), female rats (FR), male mice (MM), and female mice (FM). The task is to classify whether a chemical is POS or NEG in MR, FR, MM, and FM in terms of carcinogenicity.

4.1.2. Preprocessing and postprocessing. We use the Chinese Postman algorithm to make the input graphs Eulerian by adding the minimum amount of weights to the existing edges of the graphs. For directed graphs, we can use them directly in our algorithm, but for undirected graphs, we consider two edges in opposite directions for each undirected edge and treat the two created opposite edges as separate variables in our LP problem.

Because our method requires edge labels, for those data sets such as enzymes that have no edge labels, we use the concatenation of the source node label and the destination node label to make a label for every edge. To make the direction of the edge irrelevant, when we are comparing the two edge labels to see whether they match, we check both the equality of label of one to the label of the other or to the reverse label of the other edge, which is obtained by reversing ordering of the source and destination nodes.

After computing the distance value between each pair of graphs, we have higher values for more distant (less similar) graphs. To prepare a normalized kernel to be used in other implemented classifiers such as SVM, we have to map initial values such that for more similar graphs we obtain higher values (1 for identical pairs). To make this transformation, we have used two simple methods, and for each data set we have used both of them and chose the one that gives us the best results during the cross validation on the training set. Then, this chosen method is used on the test set to get the final accuracy. The first method is to use $f(x)=\frac{1}{x+1}$ as the map function. The second method is to use the function $f(x)=1-\frac{x-min}{max-min}$ to map the distance values. Here, the *max* and *min* show the maximum and minimum distance values that we have among all possible pairs of graphs. Since we get 0 for identical graphs, the *min* is always 0. Hence, the map function can be simplified to $f(x)=1-\frac{x}{max}$. Both methods will give us 1 for similar graphs that have GTED values of 0, and numbers between 0 and 1 for more distant graphs. The more distant the pair of graphs are, the less the corresponding value in the kernel will be.

4.1.3. Results. To evaluate whether this method works well at capturing the similarity and classifying the graphs, we used some benchmark data sets, Mutag from Debnath et al. (1991), Enzymes from Borgwardt et al. (2005), NCI1 from Wale et al. (2008), and PTC from Toivonen et al. (2003), that are used to compare the graph kernels. We compare the kernels by evaluating the accuracy of an SVM classifier that uses them for classification. We used the same settings as in Kondor and Pan (2016) so we can compare our results with previously computed results for other kernels, Weisfeiler–Lehman (WL) from Shervashidze et al. (2011), WL-Edge from Shervashidze et al. (2009), shortest path (SP) from Borgwardt and Kriegel (2005), Graphlet from Shervashidze et al. (2009), *p*-Random Walk (*p*-RW) from Gärtner (2002), and Multiscale Laplacian Graph kernel (MLG) from Kondor and Pan (2016). In this setting, we split the data randomly to two parts, 80% for training and 20% for testing. Then, we computed results for 20 different splitting using different random seeds. It can be seen from Table 1 that for the Mutag and Enzymes data sets, our kernel outperforms the other kernels. In the results (Table 1), we copied the values from Kondor and Pan (2016) for other kernels.

Table 1. Comparing GTED with Well-Known Graph Kernals

Kernel/data set	*Mutag*	*Enzymes*	*NCI1*	*PTC*
WL	84.50 (±2.16)	53.75 (±1.37)	**84.76** (±0.32)	59.97 (±1.60)
WL-Edge	82.94 (±2.33)	52.00 (±0.72)	84.65 (±0.25)	60.18 (±2.19)
SP	85.50 (±2.50)	42.31 (±1.37)	73.61 (±0.36)	59.53 (±1.71)
Graphlet	82.44 (±1.29)	30.95 (±0.73)	62.40 (±0.27)	55.88 (±0.31)
p-RW	80.33 (±1.35)	28.17 (±0.76)	TIMED OUT	59.85 (±0.95)
MLG	84.21 (±2.61)	57.92 (±5.39)	80.83 (±1.29)	**63.62** (±4.69)
GTED	**90.12** (±4.48)	**59.66** (±1.84)	65.83 (±1.14)	59.08 (±2.11)

GTED, graph traversal edit distance; MLG, Multiscale Laplacian Graph kernel; *p*-RW, *p*-Random Walk; SP, shortest path; WL, Weisfeiler–Lehman. The best accuracy on each dataset is shown in bold.

4.1.4. Analysis. As shown in Table 1, our kernel achieves a higher accuracy on the Mutag and Enzymes data sets but gets average result on PTC and relatively weaker result on NCI1, compared with other methods. Actually, none of the existing kernels can get the best results on all different kinds of data because each kernel captures only some features of the graphs. The Eulerian traversals of the graphs can be very informative for some specific applications, such as Mutag. The aromatic and heteroaromatic chemical compounds in Mutag mostly consist of connected rings of atoms. These constituent rings can give us a good measure of proximity of two compounds. Since the language of Eulerian traversals includes the traversal of these rings in each compound, finding the minimum distance between the strings of the languages (which are built by the labels of the nodes that represent the name of atoms) for two different compounds can provide a measure of the similar structures that they contain. That is why we get the best result for this data set using our kernel.

Similarly, GTED outperforms the other kernels in the enzymes data set. The enzymes in this data set have certain shapes consisting of various protein structures (the nodes), and the combination of the individual structures and the nearby proteins gives us a good sense of the structure of the enzyme. In this case, Eulerian cycles usually give us a good approximation for the general spatial structure of the enzyme, which leads to a good score.

The algorithm performed less well on the NCI1 and PTC data sets. We are uncertain of why this is, but it seems likely that the critical properties of the relevant chemicals are not captured by the Eulerian traversal.

4.2. Using GTED on genomic data

As mentioned earlier, the original goal of GTED was to find the best alignment of two genomes using only the assembly graphs, without having to create an assembled sequence first. The common alignment methods that compute the Levenshtein edit distance cannot take many factors into account, like having trans-locations in the genome, or the fact that assembly graphs could have multiple Eulerian cycles. Our method finds the best alignment among all possible alignments for all possible pairs of reference genomes that can be derived from the assembly graphs. As a result, it gives us a good measure to compute the distance (or similarity) between genomic sequences, and hence, a way to cluster a group of samples. Therefore, to evaluate our method on genomic data, we chose genomes of hepatitis B viruses in five different vertebrates; the virus in two of them (Heron and Tinamou) belong to *Avihepadnavirus* genus, and the ones in three of them (Horseshoe bat, Tent-making bat, and Woolly monkey) belong to *Orthohepadnavirus* genus.

4.2.1. Preprocessing and postprocessing. First, for each pair of sequences, we wished to compare, we generated a *colored de Bruijn graph*, a de Bruijn graph (assembly graph) that combines multiple samples in a single assembly graph with k-mers from different samples identified using different colored edges. We then extracted the graph for each specific color (genome). The LP problem for this experiment is produced almost like before; the difference here is that instead of using the second set of constraints to enforce that all edges of the input graphs are used exactly as many times as their multiplicities (an Eulerian cycle), we add the absolute value of the difference of the number of times that an edge is used in the alignment graph and its original weight in the corresponding input graph to the objective function of the LP. This way, we try to minimize this difference but allow some discrepancies. The extra flexibility seems necessary in this case because the input graphs are large and contain numerous sources of error: sequencing errors, using cutoffs for edges, and crude estimates of the weights of the edges based on the coverage of sequences in the colored de Bruijn graph mean that the edge multiplicities are not completely accurate.

4.2.2. Results. Numbers in Table 2 are the computed distance of each of these pairs of graphs. As represented in Table 2, it can be seen that the intra-genus distances are lower than inter-genus distances. We believe, based on these numbers, a good estimate of the similarity of the genomes can be made, both for genomes in the same genus and the ones with various genus.

5. EXTENSIONS

As in the problem of finding optimal alignment between two strings where the notion of local alignment and fit alignment are essential to deal with different problems, for the alignment of two graphs, these notions can become useful to solve problems that differ from what the original GTED is designed for. In

Table 2. GTED as a Distance Measure for Clustering

	Heron	*Tinamou*	*Horseshoe bat*	*Tent-making bat*	*Woolly monkey*
Heron	—	1016	1691	1639	1659
Tinamou	1016	—	1699	1638	1640
Horseshoe bat	1691	1699	—	1347	1296
Tent-making bat	1639	1638	1347	—	1429
Woolly monkey	1659	1640	1296	1429	—

The intra-genus values are shaded in gray.

this section, we demonstrate the modifications that have to be made to the original GTED to derive the analogous algorithms for graphs as presented in Ebrahimpour Boroojeny (2019).

5.1. Local GTED

When analyzing genomic sequences, we might be interested in finding a conserved section in two different genes from the same or different species. Hence, we define a variant of the original GTED, the problem of Local GTED, which is analogous to the problem of local alignment for strings.

Problem 2 (Local GTED). *Let A_1 and A_2 be two edge-labeled Eulerian graphs. We define the local edit distance between A_1 and A_2 by*

$$d_l(A_1, A_2) = \min_{\substack{s_1 \in \mathcal{C}(A_1) \\ s_2 \in \mathcal{C}(A_2)}} d(s_1, s_2), \tag{12}$$

where $\mathcal{C}(A_i)$ is the set of corresponding strings for all simple cycles in A_i for $i = 1, 2$, and $d(s_1, s_2)$ is the Levenshtein edit distance between two strings s_1 and s_2 (Levenshtein, 1966).

To solve this problem using the solution to the original GTED, we should notice that the second set of constraints does not have to be fulfilled anymore. This means that the projection of the cyclic traversal of the alignment graph onto the constituent graphs no more needs to exhaust their edges. Therefore, removing the second set of constraints provides us with an optimum traversal in $\mathcal{AG}$ whose projections to A_1 and A_2 are cyclic traversals; this will be the best local alignment that we have between A_1 and A_2.

For local GTED, something that has to be noted is the necessity of assigning a negative value for the score of a match; otherwise, the solution to the LP will be an empty traversal, which has the optimum value of 0 (minimum possible distance). By assigning a negative value to a match, we encourage the local alignment to find larger cycles in the two graphs, which represent more similar strings.

5.2. Fit GTED

There are other problems where we have to find a small subgraph of a larger graph that is similar to a smaller one. In the following, analogous to the problem of fit alignment for strings, we define a variant of the original GTED, the problem of Fit GTED.

Problem 3 (Fit GTED). *Let A_1 and A_2 be two edge-labeled Eulerian graphs. We define the local edit distance between A_1 and A_2 by*

$$d_f(A_1, A_2) = \min_{\substack{s \in \mathcal{C}(A_1) \\ \omega \in \mathcal{L}(A_2)}} d(s, \omega), \tag{13}$$

in which $\mathcal{C}(A_1)$ is the set of corresponding strings for all cycles in A_1 and $d(s, \omega)$ is the Levenshtein edit distance between two strings s and ω (Levenshtein, 1966).

For solving this problem, again we are going to use the solution to the original GTED. Here, we have to exhaust all the edges of A_2, but we can use only a subset of edges of A_1, which forms a cycle. So, by ignoring the projection constraints only for A_1, the optimum alignment that will be found by the LP will be the solution to this problem. However, this solution requires A_2 to be an Eulerian graph, and for general applications, this might not be the case. To solve this problem, we have to use the techniques discussed in

Ebrahimpour Boroojeny (2019) to relax the projection constraints for A_2 and try to minimize the distance while trying to use as many edges of A_2 as possible in the alignment.

6. CONCLUSION

In this article, we have introduced GTED, a new method for comparing networks based on a traversal of their edge labels. We have shown that GTED admits a polynomial time algorithm using a linear program. This linear program is guaranteed to have an integer solution due to the fact that the boundary operator function is totally unimodular, giving us an exact solution for the minimum possible edit distance.

The GTED problem was originally designed to be a formalization of the differential genome assembly problem, comparing DNA assembly graphs by considering all their possible assembled strings. It performs well at that task, successfully differentiating different genera of the hepatitis B virus. We tested GTED on viral genomes since GTED is a global alignment scheme that is not immediately scalable to full-size large genomes, like all other global alignment schemes such as Needleman–Wunsch. However, GTED can form the mathematical basis for scalable heuristic comparison of full-size large genomes in the future. GTED can also be used as a general graph kernel on other types of networks, performing particularly well on graphs whose Eulerian traversals provide a good insight into their important structural features.

GTED is a new way of measuring the similarity between networks. It has many applications in differential genome assembly, but it also performs well in domains beyond assembly graphs. GTED has the potential to be a valuable tool in the study of biological networks.

AUTHOR DISCLOSURE STATEMENT

The authors declare they have no competing financial interests.

FUNDING INFORMATION

No funding was received for this project.

REFERENCES

Borgwardt, K.M., and Kriegel, H.P. 2005. Shortest-path kernels on graphs. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, Houston, TX. p. 8.

Borgwardt, K.M., Ong, C.S., Schönauer, S., et al. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, 47–56.

Boroojeny, A.E., Shrestha, A., Sharifi-Zarchi, A., et al. 2018. GTED: Graph traversal edit distance. In *International Conference on Research in Computational Molecular Biology*. Springer, Paris, France. pp. 37–53.

Debnath, A.K., Lopez de Compadre, R.L., Debnath, G., et al. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *J. Med. Chem.* 34, 786–797.

Dey, T., Hirani, A., and Krishnamoorthy, B. 2011. Optimal homologous cycles, total unimodularity, and linear programming. *SIAM J. Comput.* 40, 1026–1044.

Ebrahimpour Boroojeny, A. 2019. Theory of graph traversal edit distance, extensions, and applications [PhD thesis]. Available from ProQuest Dissertations & Theses Global database, Ann Arbor, MI. (Accession Order No. AAT 13812734).

Ebrahimpour Boroojeny, A., Shrestha, A., Sharifi-Zarchi, A., et al. 2020. PyGTED: Python application for computing graph traversal edit distance. *J. Comp. Biol.* 27 (this issue).

Feragen, A., Kasenburg, N., Petersen, J., et al. 2013. Scalable kernels for graphs with continuous attributes, 216–224. *In* Burges, C.J.C., Bottou, L., Welling, M., et al., eds., *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., Lake Tahoe, CA.

Gärtner, T. 2002. Exponential and geometric kernels for graphs. In *NIPS 2002 Workshop on Unreal Data*. Principles of Modeling Nonvectorial Data, Vancouver, British Columbia.

Hormozdiari, F., Hajirasouliha, I., McPherson, A., et al. 2011. Simultaneous structural variation discovery among multiple paired-end sequenced genomes. *Genome Res.* 21, 2203–2212.

Inokuchi, A., Washio, T., and Motoda, H. 2003. Complete mining of frequent patterns from graphs: Mining graph data. *Mach Learn.* 50, 321–354.

Iqbal, Z., Caccamo, M., Turner, I., et al. 2012. De novo assembly and genotyping of variants using colored de bruijn graphs. *Nat Genet.* 44, 226–232.

Jones, N.C., and Pevzner, P. 2004. *An Introduction to Bioinformatics Algorithms*. MIT Press, Cambridge, MA.

Jones, S. 2012. True colors of genome variation (research highlights). *Nat. Biotechnol* 30, 158.

Kondor, R., and Borgwardt, K.M. 2008. The skew spectrum of graphs. In *Proceedings of the 25th International Conference on Machine Learning*. ICML'08. ACM, New York, NY. pp. 496–503.

Kondor, R., and Pan, H. 2016. The multiscale laplacian graph kernel, 2990–2998. *In* Lee, D.D., Sugiyama, M., Luxburg,U.V., Guyon, I., and Garnett, R., eds., *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc.

Kubinyi, H. 2003. Drug research: Myths, hype and reality. *Nat Rev Drug Discov.* 2, 665–668.

Levenshtein, V.I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics-Doklady* 10, 707–710. Original in *Doklady Akademii Nauk SSSR* 163:845–848 (1965).

Li Y, Zheng H, Luo R, et al. 2011. Structural variation in two human genomes mapped at single-nucleotide resolution by whole genome de novo assembly. *Nat Biotechnol.* 29, 723–730.

Mak, C. 2011. Multigenome analysis of variation (research highlights). *Nat Biotechnol.* 29, 1101.

Massey, W. 1991. *A Basic Course in Algebraic Topology*. Volume 127. Springer, Berlin, Germany.

Movahedi, N.S., Embree, M., Nagarajan, H., et al. 2016. Efficient synergistic single-cell genome assembly. *Front Bioeng Biotechnol.* 4, 42.

Movahedi, N.S., Forouzmand, E., and Chitsaz, H. 2012. De novo co-assembly of bacterial genomes from multiple single cells. In Tian, T. (ed.): *BIBM*, IEEE, Philadelphia, PA. pp. 561–565.

Myers, E.W. 1995. Toward simplifying and accurately formulating fragment assembly. *J Comput Biol.* 2, 275–290.

Neumann, M., Garnett, R., Bauckhage, C., et al. 2016. Propagation kernels: Efficient graph kernels from propagated information. *Mach Learn.* 102, 209–245.

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

Pevzner, P.A., Tang, H., and Waterman, M.S. 2001. An Eulerian path approach to DNA fragment assembly. *Proc Natl Acad Sci USA.* 98, 9748–9753.

Ronen, R., Boucher, C., Chitsaz, H., et al. 2012. SEQuel: Improving the accuracy of genome assemblies. *Bioinformatics.* 28, 12, i188–i196. also ISMB proceedings.

Shervashidze, N., Schweitzer, P., van Leeuwen, E.J., et al. 2011. Weisfeiler-lehman graph kernels. *J Mach Learn Res.* 12, 2539–2561.

Shervashidze, N., Vishwanathan, S., Petri, T., et al. 2009. Efficient graphlet kernels for large graph comparison, 488–495. *In* van Dyk, D. and Welling, M., eds., *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*. Volume 5 of *Proceedings of Machine Learning Research*. PMLR, Hilton Clearwater Beach Resort, Clearwater Beach, FL.

Simpson, J.T., and Durbin, R. 2010. Efficient construction of an assembly string graph using the FM-index. *Bioinformatics.* 26, 367–373.

Taghavi, Z., Movahedi, N.S., Draghici, S., et al. 2013. Distilled single-cell genome sequencing and de novo assembly for sparse microbial communities. *Bioinformatics.* 29, 2395–2401.

Toivonen, H., Srinivasan, A., King, R.D., et al. 2003. Statistical evaluation of the predictive toxicology challenge 2000–2001. *Bioinformatics.* 19, 1183–1193.

Tutte, W., and Smith, C. 1941. On unicursal paths in a network of degree 4. *Am Math Monthly.* 48, 233–237.

van Aardenne-Ehrenfest, T., and de Bruijn, N. 1987. Circuits and trees in oriented linear graphs, 149–163. *In* Gessel, I. and Rota, G.C., eds., *Classic Papers in Combinatorics*. Modern Birkhäuser Classics. Birkhäuser Boston.

Vick, J.W. 1994. *Homology Theory: An Introduction to Algebraic Topology*. Vol. 145. Springer, Berlin, Germany.

Vishwanathan, S.V.N., Schraudolph, N.N., Kondor, R., et al. 2010. Graph kernels. *J Mach Learn Res.* 11, 1201–1242.

Wale, N., Watson, I.A., and Karypis, G. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowl Inf Syst.* 14, 347–375.

Address correspondence to:
Dr. Hamidreza Chitsaz
Department of Computer Science
Colorado State University
279 Computer Science Building
1873 Campus Delivery
Fort Collins, CO 80523-1873

E-mail: chitsaz@chitsazlab.org