Alignment Kernels Based on a Generalization of Alignments

Kilho SHIN\(^{(a)}\), Member

SUMMARY This paper shows a way to derive positive definite kernels from edit distances. It is well-known that, if a distance \(d\) is negative definite, \(e^{-\lambda d}\) is positive definite for any \(\lambda > 0\). This property provides us the opportunity to apply useful techniques of kernel multivariate analysis to the features of data captured by means of the distance. However, the known instances of edit distance are not always negative definite. Even worse, it is usually not easy to examine whether a given instance of edit distance is negative definite. This paper introduces alignment kernels to present an alternative means to derive kernels from edit distance. The most important advantage of the alignment kernel consists in its easy-to-check sufficient condition for the positive definiteness. In fact, when we surveyed edit distances for strings, trees and graphs, all but one are instantly verified to meet the condition and therefore proven to be positive definite.

**key words:** edit distance, kernel, tree, graph

1. Introduction

Embedding of metric spaces into a Hilbert space has been an important problem of mathematics, and the notion of negative definite kernel (for definition, refer to [1]) was introduced in the course of solving this problem. The problem was solved when Schoenberg ([2]) proved that, if a negative definite kernel \(\psi: \mathcal{W} \times \mathcal{W} \to \mathbb{R}\) satisfies \(\psi(x, x) = 0\) for all \(x \in \mathcal{W}\), there exist a Hilbert space \(\mathcal{H}\) and a mapping \(\epsilon: \mathcal{W} \to \mathcal{H}\) such that \(\psi(x, y) = \|\epsilon(x) - \epsilon(y)\|_2^2\).

Moreover, Schoenberg proved Theorem 1, which characterizes negative definite kernels in a different way.

**Theorem 1** ([2]): For \(\psi: \mathcal{W} \times \mathcal{W} \to \mathbb{C}\), \(\psi\) is negative definite, if, and only if, \(e^{-\lambda \psi(x,y)}\) is positive definite for any \(\lambda > 0\).

This theorem is practically important as well. If \(e^{-\lambda \psi(x,y)}\) is positive definite (for definition, refer to [1]), there exists another embedding \(\epsilon': \mathcal{W} \to \mathcal{H}\) such that \(\psi(x, y) = \langle \epsilon'(x), \epsilon'(y) \rangle\) (reproducing kernel Hilbert space). We let \(\langle \cdot, \cdot \rangle\) denote the inner product associated with \(\mathcal{H}\). Because many multivariate analysis techniques can be performed only based on the inner products of data, we can take advantage of these techniques without the knowledge of \(\epsilon'\).

This methodology is known as kernel multivariate analysis.

Edit distance, on the other hand, has been widely used to study discrete structures. Barring our expectation, edit distances for strings, trees and graphs, all but one are instantly verified to meet the condition and therefore proven to be positive definite. Cortes et al. showed this for Levenshtein distance, an important instance of edit distance for strings ([13]); Also, it follows from our notion of alignment graphs that other major instances of edit distance for trees and graphs are not negative definite, either (Appendix B). Therefore, in order to leverage structural information of data carried by edit distances by means of multivariate analysis, we had to invent indirect methods to derive kernels from edit distances. We see two examples of such methods. First, the Lipschitz embedding is an interesting method to embed data into a Hilbert space. We first sample a relatively small number of data from a dataset (basing points) and then determine the coordinates of each datum by the edit distances of the datum from the basing points ([4]). Although the Lipschitz embedding indeed projects data into a Hilbert space, its intrinsic meaning is unclear. Secondly, if \(d(x, y)^2 = \|\epsilon(x) - \epsilon(y)\|_2^2\) would follow. Hence, it is a reasonable attempt to define \(K(x, y)\) by the right-hand side of this equality, although such \(K(x, y)\) is not always positive definite. In [5], an interesting method to make a sum or a product of \(K(x, y)\) over \(z \in I\) is introduced.

In this regard, this paper presents a direct method to derive positive definite kernels based on the edit distance methodology. In our method, we approximate the edit distance \(d: \mathcal{W} \times \mathcal{W} \to [0, \infty)\) by a certain family of distances \(\beta(x, y) = d(x, y)^2 + \lambda d(x, y)^2\) for all \(x, y \in \mathcal{W}\) and \(\lambda > 0\). Finally, we define alignment kernels \(K^\lambda(x, y) = e^{-\lambda \beta(x, y)}\). Although \(K^\lambda\) are not necessarily positive definite, an easy-to-check sufficient condition for positive definiteness exists. Moreover, from our survey, we can expect that the condition holds true for many edit distance definitions.

To define \(\beta\), we like to take as generic an approach as possible. Hence, we assume the abstract model of edit distances that suppose only the following.

1. An object in the domain \(\mathcal{W}\) consists of a finite graph and arbitrary additional structural information: For example, a string is a linear graph and an unrooted tree is an acyclic connected graph; By adding the generation (ancestor-descendant) order of vertices to an unrooted tree, we obtain a rooted unordered tree; Furthermore, an ordered tree is accompanied by the additional structural information of the sibling (left-to-right) order.
2. An edit path consists of edit operations, which fall into one of these types: (i) deleting a vertex or an edge,
(ii) inserting a vertex or an edge and (iii) substituting a new label for the label of a vertex or an edge. If we obtain \(y \in W\) by sequentially applying the edit path to \(x \in W\), we say that the path transforms \(x\) into \(y\).

Based on this abstract model of edit distances, we first introduce the notion of alignment, and then, based on the notion, we define \(d^A\). We will illustrate this method taking Levenshtein distance for an instance. Assume that we are given two strings \textit{exodus} and \textit{exorcism}. An example of the alignments between these strings is given as follows.

\[
e \ x \ o \ o \bullet d \ u \bullet s \bullet \\
e \ x \ o \ r \ c \bullet i \ s \ m
\]

(1)

Each column of the alignment represents a single operation: \(\langle \xi, \eta \rangle\) and \(\langle \cdot, \cdot \rangle\) represent the operations of deletion, insertion and substitution, respectively. Thus, the alignment uniquely determines an edit path. Therefore, when we let \(A_{x,y}\) denote the set of alignments between \(x = \textit{exodus}\) and \(y = \textit{exorcism}\) and let \(\gamma\) denote the cost function for edit paths, Levenshtein distance \(d(x, y)\) is given by \(d(x, y) = \min \gamma(\alpha)\).

This formulation of \(d(x, y)\) leads us to the definition of

\[
\mathcal{d}^A(x, y) = -\frac{1}{\lambda} \log \left( \sum_{\alpha \in A_{x,y}} e^{-\lambda \gamma(\alpha)} \right).
\]

By making use of the soft minimum approximation of

\[
0 \leq \min\{a_1, \ldots, a_n\} + \frac{1}{\lambda} \log \left( \sum_{i} e^{-a_i} \right) \leq \frac{\log n}{\lambda},
\]

we have \(0 \leq d(x, y) - \mathcal{d}^A(x, y) < \frac{(|x| + |y|) \log^2 2}{\lambda} \cdot |A_{x,y}| < 2|x+y|\), and hence \(\lim d(x, y) = d(x, y)\) holds true. Moreover, to prove that \(e^{-\lambda \gamma(x,y)}\) is positive definite, we can leverage the theory of mapping kernels presented in [6].

To abstract this example, we need a generic definition of alignments, but Levenshtein and less-constrained distances are the only examples in the literature for which the notion of alignments is defined.

Thus, the first contribution of this paper is to present a generic definition of alignments based on the aforementioned abstract model of edit distances. Although our definition is different from the definitions presented in the literature for Levenshtein and less-constrained distances, these legacy definitions turn out to be derived from our definition.

The second contribution of this paper is to show a sufficient condition for the resulting alignment kernels to be positive definite (Theorem 4). The condition is not only easy-to-check but also likely to hold true for many edit distance definitions, since the triangle inequality of edit distances, if it holds, is usually derived from a certain stronger condition. For example, among the six edit distance definitions that we surveyed, only the less-constrained edit distance for trees does not meet the condition. In fact, we can determine a trilet of trees such that the Gram matrix for them with respect to the less-constrained edit distance is not positive definite and the triangle inequality does not hold (4.4).

Lastly, this paper is based on the paper that the author presented at International Conference on Artificial Intelligence, Soft Computing 2013 ([7]).

2. Notations

A graph means an undirected attributed graph. A graph \(x\) is a four-tuple \((V_x, E_x, \ell^V_x, \ell^E_x)\), where \(V_x\) and \(E_x\) are finite sets of vertices and edges, and \(\ell^V_x : V_x \rightarrow \Sigma\) and \(\ell^E_x : E_x \rightarrow \Sigma\) are functions to determine labels. Although we assume the alphabet \(\Sigma\) includes the null label \(\bullet\), unless the graph is an alignment graph (see Sect. 3), \(\bullet\) cannot be used.

In the following, we let \(W\) denote a domain of graphs, and assume \(x, y \in W\).

A graph homomorphism \(f : x \rightarrow y\) is a pair \((f^V : V_x \rightarrow V_y, f^E : E_x \rightarrow E_y)\) such that \(f^E((v, \ell^V_x(v))) = (f^V(v), \ell^V_y(f^V(v)))\) always holds, whereas a partial graph homomorphism is a graph homomorphism from a subgraph of \(x\) to \(y\).

A graph homeomorphism is a graph homomorphism such that \(\ell^V\) and \(\ell^E\) are bijective, whereas a subgraph homeomorphism is a graph homomorphism such that \(\ell^V\) and \(\ell^E\) are injective.

A subgraph isomorphism is a subgraph homomorphism such that \(\ell^V_y \circ f^V = \ell^V_x\) and \(\ell^E_y \circ f^E = \ell^E_x\) always hold, whereas a graph isomorphism is a subgraph isomorphism such that \(f^V\) and \(f^E\) are bijective.

An edit operation of graphs falls into one of deletion, insertion and substitution of vertices or edges. We let \(\langle \xi, \bullet \rangle\) and \(\langle \bullet, \xi \rangle\) denote the deletion and insertion of \(\xi\) and \(\langle \xi, \eta \rangle\) do the substitution from \(\xi\) to \(\eta\), and \(\langle \xi, \xi \rangle\) denotes the null label.

A cost function \(\gamma : \Sigma \times \Sigma \rightarrow \mathbb{R}\) determines costs of edit operations. For convenience of explanation, we often leave out the symbols of label functions. For example, \((v, w) \in V_x \times V_y\), \(\eta(v, w)\) indicates \(\gamma(\ell^V_x(v), \ell^V_y(w))\), and means the cost of \((v, w)\). We require that \(\gamma\) is commutative, that is, \(\gamma(\xi, \eta) = \gamma(\eta, \xi)\), and satisfies the triangle inequality, that is, \(\gamma(\xi, \zeta) \leq \gamma(\xi, \eta) + \gamma(\eta, \zeta)\).

An edit path is a sequence of edit operations that converts \(x\) into \(y\). The cost of an edit path is defined as the sum of the costs of the edit operations that the path comprises, and the edit distance \(d_W(x, y)\) is the minimum of the costs across all the possible edit paths that transform \(x\) into \(y\).

3. Alignment Graphs

The aim of this section is to define the set \(A_{x,y}^{dw}\) of alignment graphs, determined per \((x, y) \in W \times W\).

3.1 Traces and Alignments – Legacy Definitions

The trace \(\tau\) of an edit path between \(x\) and \(y\) is defined as the one-to-one partial mapping that maps vertices \(v\) and edges \(e\) of \(x\) to vertices and edges of \(y\) so that \((v, \tau(v))\) and \((e, \tau(e))\) exactly determine the substituting operations of the path.
For example, the trace of the edit path determined by Diagram (1) is \(\{(e, e), (x, x), (o, o), (d, c), (s, s)\}\).

Note that Levenshtein distance for strings, and Tai, Lu, the constrained and the less-constrained distances for trees do not assume labels of edges, and implicitly assume that all the edges are deleted and then inserted with cost zero. Therefore, the trace of an edit path from \(x\) to \(y\) for these edit distance definitions is a partial one-to-one mapping from \(V_x\) to \(V_y\). On the other hand, a one-to-one partial mapping \(\tau: V_x \rightarrow V_y\) is not always a trace for some edit path. The condition that \(\tau\) becomes a trace varies dependent on each edit distance definition as follows.

- **Levenshtein:** \(\tau(v)\) is left to \(\tau(v')\), if \(v\) is left to \(v'\).
- **Tai:** \(\tau(v)\) is left to \(\tau(v')\), if \(v\) is left to \(v'\); \(\tau(v)\) is an ancestor of \(\tau(v')\), if \(v\) is an ancestor of \(v'\).

For Lu, the constrained, and the less-constrained distances, the following conditions are imposed in addition to the conditions for Tai distance. \(v_1 \sim v_j\) denotes the nearest common ancestor of \(v_i\) and \(v_j\).

- **Lu:** \(\tau(v_1) \sim \tau(v_2)\) is an ancestor of \(\tau(v_1) \sim \tau(v_2)\), if \(v_1 \sim v_2\) is an ancestor of \(v_1 \sim v_2\).
- **Constrained:** The same condition as Lu’s case holds, only when \(v_1\) is not an ancestor of the others.
- **Less-Constrained:** No twist (see below) is included.

The dotted arrows determine a trace between two trees.

The corresponding alignment tree is given as a dual-label tree. Tracing the left-hand (right-hand) labels yields the left (right) tree.

**Fig. 1** An alignment for the less-constrained tree edit distance.

---

**Twist.** The vertices displayed as a square are the nearest common ancestors of vertices \(v_i\)’s or \(w_i = \tau(v_i)\)’s.

Edit distance definitions for general graphs, by contrast, handle labels of edges explicitly, and therefore, the trace of an edit path is a pair of partial one-to-one mappings \(\tau^V: V_x \rightarrow V_y\) and \(\tau^E: E_x \rightarrow E_y\).

The most common edit distance definition for graphs known in the literature requires that a vertex can be deleted after deleting all the edges connected to it (we call it the generic graph edit distance). Hence, the necessary and sufficient condition for the pair \((\tau^V, \tau^E)\) to be a trace is \(\tau^E(v_1, v_2) = (w_1, w_2) \Rightarrow \tau^V(v_1) = w_1 \wedge \tau^V(v_2) = w_2\).

By contrast with the trace, the notion of alignments is not clearly defined for every edit distance definition: For Levenshtein distance, an alignment is usually defined as a pair of strings as depicted by Diagram (1); For the less-constrained distance, an alignment is defined as a tree with dual-labels as depicted by Fig. 1; To our knowledge, no formal definition of alignment is given to Tai, Lu and the constrained distances for trees.

### 3.2 Trace Mappings and Alignment Graphs

Definition 1 gives a formal description of traces, and Definition 4 generalizes the notion of alignments so that it generically applies to a wide range of objects. We let \(d_W: \mathcal{W} \times \mathcal{W} \rightarrow [0, \infty)\) be an edit distance function.

**Definition 1:** For an edit path that transforms \(x\) into \(y\), the trace mapping of the edit path is the partial subgraph homeomorphism from \(x\) to \(y\) that maps \(\xi \in V_x \cup E_x\) to \(\eta \in V_y \cup E_y\), if, and only if, the edit path substitutes \(\eta\) for \(\xi\).

**Definition 2:** For \((x, y) \in \mathcal{W} \times \mathcal{W}\), \(\mathcal{W}^d_{xy}\) is the set of trace mappings of all the edit paths from \(x\) to \(y\), determined associated with the distance \(d_W\).

**Definition 3:** A benign extension \((\bar{x}, \epsilon_x)\) of \(x \in \mathcal{W}\) is a subgraph isomorphism \(\epsilon_x\) from \(x\) to \(\bar{x}\) such that \(\ell^V_x(V_x \setminus \epsilon_x(V_x)) = \ell_y^V(E_x \setminus \epsilon_x(E_x)) = \emptyset\). The infinite set of all benign extensions of \(x\) is denoted by \(\mathcal{B}_x\).

**Definition 4:** For \(x, y \in \mathcal{W}\), an alignment graph from \(x\) to \(y\) is a graph homeomorphism \(\iota\) from \(\bar{x}\) of some \((\bar{x}, \epsilon_x) \in \mathcal{B}_x\) to \(\bar{y}\) of some \((\bar{y}, \epsilon_y) \in \mathcal{B}_y\) that meet the following conditions.

1. For all \(v \in V_x\), if \(\ell^V_x(v) = \emptyset\), then \(\ell^V_x(\iota(v)) \neq \emptyset\).
2. For all \(e \in E_x\), if \(\ell^E_x(e) = \emptyset\), then \(\ell^E_y(\iota(\epsilon_x(e))) \neq \emptyset\).
3. \((\epsilon_y)^{-1} \circ \iota \circ \epsilon_x \in \mathcal{T}_{w_{xy}}\).

The partial subgraph homeomorphism \((\epsilon_y)^{-1} \circ \iota \circ \epsilon_x\) uniquely determines the alignment graph \(\iota\) up to isomorphism.

**Lemma 1:** Let \(\iota\) and \(\iota'\) be alignment graphs from \(x\) to \(y\), \(\iota\) is from \((\bar{x}, \epsilon_x) \in \mathcal{B}_x\) to \((\bar{y}, \epsilon_y) \in \mathcal{B}_y\), and \(\iota'\) is from \((\bar{x}', \epsilon_x') \in \mathcal{B}_x\) to \((\bar{y}', \epsilon_y') \in \mathcal{B}_y\). If \((\epsilon_y)^{-1} \circ \iota \circ \epsilon_x\) is identical to \((\epsilon_y')^{-1} \circ \iota' \circ \epsilon_x'\), graph isomorphisms \(\alpha_x: \bar{x} \rightarrow \bar{x}'\) and \(\alpha_y: \bar{y} \rightarrow \bar{y}'\) exist, and the following commutative diagram holds.

**Proof.** First, we define \(\alpha_x^V: V_x \rightarrow V_{\bar{x}}\) as follows.

\[\alpha_x^V(v) = \begin{cases} \epsilon_x^V(v), & \text{if } v = \epsilon_x^V(v), \\ (\iota')^{-1} \circ \epsilon_y^V(v), & \text{if } v = (\iota')^{-1} \circ \epsilon_y^V(v). \end{cases}\]

On one hand, \(V_x = \mathcal{V}(\epsilon_x') \cup \mathcal{V}(\epsilon_y)^{-1} \circ \iota' \circ \epsilon_x'\) holds by definition. On the other hand, if \(v = \epsilon_x^V(v) = (\iota')^{-1} \circ \epsilon_y^V(v)\), \(e_y^V(v) = (\iota')^{-1} \circ \epsilon_y^V(v)\) follows from the hypothesis.
\((\varepsilon'_g)^{-1} \circ \ell'^V \circ \varepsilon'_x = (\varepsilon'_g)^{-1} \circ \ell'^V \circ \varepsilon'_x\). Thus, \(\alpha_x^V\) is well defined. It is evident that \(\alpha_x^V\) is bijective and preserves labels. In the same way, we can define \(\alpha_x^E\), and can show that it is bijective and preserves labels.

Thus, \(\alpha_x = (\alpha_x^V, \alpha_x^E)\) is a graph isomorphism.

When we define \(\alpha_y\) exactly in the same way as \(\alpha_x\), it is a graph isomorphism again. The claimed commutative diagram follows from the definitions of \(\alpha_x\) and \(\alpha_y\).

Lemma 2: For a trace mapping \(\tau \in T_{x,y}^{dw}\), there exists an alignment \(\iota \in A_{x,y}^{dw}\) such that \(\tau = (\varepsilon_g)^{-1} \circ \iota \circ \varepsilon_x\).

Proof. To determine \((\bar{x}, \varepsilon_x) \in B_x\), we first describe a graph \(g\) as follows.

1. We define a graph \(\bar{g}\) by \(V_{\bar{g}} = V_x \sqcup V_y, E_{\bar{g}} = E_x \sqcup E_y\) and \(3 \mathfrak{m}(\varepsilon'_x) \cup \mathfrak{m}(\varepsilon'_y) = \{\bullet\}\). \(X \sqcup Y\) means the disjoint union of two sets \(X\) and \(Y\).
2. For every \((v, v') \in V_{\bar{g}} \times V_{\bar{g}}\), if \(v' = \tau^V(v)\), we unify \(v\) and \(v'\) into a single vertex.
3. If two edges connect the same pair of vertices, and if \(\tau^E\) maps one to the other, we unify these edges into a single edge.

Furthermore, we convert the resulting graph \(g\) into \(\bar{x}\) by relabeling \(V_{\bar{g}}\) and \(E_{\bar{g}}\) so that the canonical subgraph homeomorphism from \(x\) to \(g\) gives a benign extension \(\bar{x}\). In the same way, we define \((\bar{y}, \varepsilon_y)\). When we derive a homeomorphism \(\iota : \bar{x} \rightarrow \bar{y}\) from the identity mapping of \(g\), \(\tau = (\varepsilon_g)^{-1} \circ \iota \circ \varepsilon_x\) evidently holds.

Lemma 1 and 2 immediately imply Theorem 2.

Theorem 2: When define \(\alpha_{dw} : A_{x,y}^{dw} \rightarrow T_{x,y}^{dw}\) by

\[
\alpha_{dw} : (\iota \rightarrow (\bar{x}, \varepsilon_x) \rightarrow (\bar{y}, \varepsilon_y)) \mapsto (\varepsilon_g)^{-1} \circ \iota \circ \varepsilon_x,
\]

\(\alpha_{dw}\) is bijective.

On the other hand, Theorem 3 presents the way how to calculate \(d_W(x, y)\) from \(A_{x,y}^{dw}\).

Theorem 3: The following calculates \(d_W(x, y)\) from \(A_{x,y}^{dw}\).

\[
d_W(x, y) = \min_{\iota \in A_{x,y}^{dw}} \left( \sum_{v \in V_x} \gamma(v, \tau^V(v)) + \sum_{e \in E_x} \gamma(e, \tau^E(e)) \right)
\]

Proof. Let \(\tau\) be a trace mapping. By the triangle inequality holding for \(\gamma\), the following gives the minimum cost for the edit paths that share \(\tau\) as the trace mapping.

\[
\sum_{v \in \mathfrak{d}(\tau^V)} \gamma(v, \tau^V(v)) + \sum_{v \in V_x \setminus \mathfrak{d}(\tau^V)} \sum_{v \in \mathfrak{d}(\tau^V)} \gamma(v, \bullet) + \sum_{v \in \mathfrak{d}(\tau^V)} \gamma(v, \bullet) + \sum_{e \in \mathfrak{d}(\tau^E)} \gamma(e, \tau^E(e)) + \sum_{e \in E_x \setminus \mathfrak{d}(\tau^E)} \sum_{e \in \mathfrak{d}(\tau^E)} \gamma(e, \bullet) + \sum_{e \in \mathfrak{d}(\tau^E)} \gamma(e, \bullet)
\]

The assertion immediately follows.

3.3 Examples

Although the edit distance definitions for strings and trees that we investigate in this paper ignore edge labels, we assume that all the edges are labeled with 1 for formalization.

(1) Levenshtein distance

The alignment graph for the trace mapping of Diagram (1) is given as follows.

\[
\begin{array}{c}
\gamma \quad \tau \quad \delta \\
\end{array}
\]

(2) The tree edit distances

The following gives the alignment graph for the trace mapping of Fig. 1.

(3) Tai tree edit distance

The following is an example of the trace mappings for Tai distance that cannot be a trace mapping for Lu, the constrained and the less-constrained distances.

The corresponding alignment graph is given as follows.
The dotted lines represent the edges labeled with $\bullet$, while the solid lines do the edges labeled with 1.

$$\begin{array}{c}
e \quad a \quad d \quad \bullet \quad e \\
| b & c & d & e & f \\
\end{array} \quad \quad \quad \begin{array}{c}
e \quad a \quad d \\
| b' & c' & d' \end{array}$$

(6) The diagram left below determines a trace mapping between two triangular graphs (the dotted arrows), while the diagram right below does the corresponding alignment graph.

4. Alignment Kernels

Based on the results of Sect. 3, we introduce alignment kernels. Although $e^{-\lambda d_W(x,y)}$ is positive definite if $d_W$ is negative definite, it is usually not easy to examine whether edit distance is negative definite. By contrast, the alignment kernel is accompanied by an easy-to-check sufficient condition for the positive definiteness.

4.1 Definitions and Main Theorems

We start with defining alignment kernels.

**Definition 5:** For a family of alignment graphs $\mathcal{A} = \{A_{x,y} \mid x, y \in \mathcal{W}\}$, the associated alignment kernel with a positive parameter $\lambda$ is defined as follows.

$$K_{A_{x,y}}^\lambda (x, y) = \sum_{e \in A_{x,y}} \prod_{v \in V_1} e^{-\lambda y(v, e)} \cdot \prod_{e \in E_1} e^{-\lambda y(e, f)}$$

To introduce this definition, we first approximate $d_W$ by applying the soft minimum approximation to the formula of Theorem 3. When we define $d_W^\lambda (x, y)$ by

$$d_W^\lambda (x, y) = \frac{1}{\lambda} \log \left( \sum_{e \in A_{x,y}} \prod_{v \in V_1} e^{-\lambda y(v, e)} \cdot \prod_{e \in E_1} e^{-\lambda y(e, f)} \right),$$

$d_W(x, y) = \lim_{\lambda \to 0} d_W^\lambda (x, y)$ holds by Theorem 3. Finally, we define the alignment kernel by $K_{A_{x,y}}^\lambda (x, y) = e^{-\lambda d_W^\lambda (x, y)}$.

**Theorem 4:** $K_{A_{x,y}}^\lambda (x, y)$ is positive definite, if the kernel $e^{-\lambda y} |_{\bar{E}(\mathcal{W})} \cdot |_{\mathcal{W}}$ is positive definite and $\alpha_{d_W}(A_{x,y}) = \{\alpha_{d_W}(A_{x,y}) \mid x, y \in \mathcal{W}\}$ is transitive.

By Theorem 2, $\alpha_{d_W}(A_{x,y})$ is identical to $\mathcal{T}_{d_W} = \{\mathcal{T}_{x,y} \mid x, y \in \mathcal{W}\}$, and the transitivity is defined as follows.

**Definition 6:** $\mathcal{T}_{d_W}$ is transitive, iff the following holds.

$$\tau \in \mathcal{T}_{x,y} \land \tau' \in \mathcal{T}_{y,z} \land \Im (\tau) = \mathcal{D}om(\tau') \implies \tau' \circ \tau \in \mathcal{T}_{x,z}.$$
is positive definite.

**Proof.** We fix $i \in A_{\lambda \gamma}^{d_i}$ and $\tau = \alpha \gamma(i) \in T_{\lambda \gamma}^{d_i}$.

$$\prod_{v \in V_{\lambda \gamma}} e^{-\lambda \gamma(v \cdot (v))}$$

$$= \prod_{v \in V_{\lambda \gamma}} e^{-\lambda \gamma(v \cdot (v))} \cdot \prod_{v \in V_{\lambda \gamma}} e^{-\lambda \gamma(v \cdot (v))}$$

$$= \prod_{v \in V_{\lambda \gamma}} e^{-\lambda \gamma(v \cdot (v))} \cdot \prod_{v \in V_{\lambda \gamma}} e^{-\lambda \gamma(v \cdot (v))}$$

$$\lambda \gamma(v \cdot (v))$$

We do the same computation for $\prod_{v \in V_{\lambda \gamma}} e^{-\lambda \gamma(v \cdot (v))}$, and then can transform $K_{\lambda \gamma d_i}^\lambda(x, y)$ as follows.

$$K_{\lambda \gamma d_i}^\lambda(x, y) = \left( \prod_{v \in V_{\lambda \gamma}} e^{-\lambda \gamma(v \cdot (v))} \cdot \prod_{v \in V_{\lambda \gamma}} e^{-\lambda \gamma(v \cdot (v))} \right)$$

The assertion follows. □

The positive definiteness of $K_{\lambda \gamma d_i}^\lambda(x, y)$ is deduced from the theory of mapping kernels ([6]), to be specific, Theorem 6. For the deduction, some combinatoric trick is necessary, but is not difficult.

**Theorem 6 ([6]):** The following conditions are equivalent.

1. ($M_{\lambda \gamma}$) is transitive.
2. $K(x, y) = \sum_{(x', y') \in M_{\lambda \gamma}} \kappa(x', y')$ is positive definite for any positive definite $\kappa$.

$M_{\lambda \gamma}$ is a subset of $X \times X$ such that the kernel $\kappa$ is defined over $X \times X$. Furthermore, we define that $M_{\lambda \gamma}$ is transitive, if, and only if, $(x', y') \in M_{\lambda \gamma} \Rightarrow (y', x') \in M_{\lambda \gamma}$ and $(x', y') \in M_{\lambda \gamma} \land (y', z') \in M_{\lambda \gamma} \Rightarrow (x', z') \in M_{\lambda \gamma}$.

### 4.3 Proof of Theorem 5

The key to our proof is the formula of $d_{\lambda \gamma}(x, y) = \lim_{t \to \infty} D_{\lambda \gamma}(x, y)$. For any $\mu > 0$, we have

$$e^{-\mu d_{\lambda \gamma}(x, y)} = e^{-\mu \lim_{t \to \infty} D_{\lambda \gamma}(x, y)}$$

$$= e^{-\mu \lim_{t \to \infty} -\frac{1}{2} \log K_{\lambda \gamma d_i}^\lambda(x, y)}$$

$$= \lim_{t \to \infty} \left( K_{\lambda \gamma d_i}^\lambda(x, y) \right)^{\mu / \lambda}$$

$$K_{\lambda \gamma d_i}^\lambda(x, y)$$

is positive definite because $K_{\lambda \gamma d_i}^\lambda$ is infinitely divisible, and hence, $e^{-\mu d_{\lambda \gamma}(x, y)}$ is positive definite. It follows from Theorem 1 that $d_{\lambda \gamma}(x, y)$ is negative definite.

### 4.4 Examples

Theorem 4 requires the positive definiteness of $e^{-\lambda \gamma} |_{\Sigma i} \times \Sigma i)$ and the transitivity of $\alpha \gamma(\lambda \gamma d_i)$ as hypotheses.

The first condition holds true for the common settings of $\gamma(\xi, \eta) = 1 - \delta_{\xi, \eta}$ and $\gamma(\xi, \eta) = ||\xi - \eta||_2$ or $||\xi - \eta||_1$ when labels are real vectors.

Also, the second condition is met for many of the edit distance definitions presented in the literature. In fact, $\alpha \gamma(\lambda \gamma d_i)$ proves to be strongly transitive for Levenshtein distance for strings, for Tai, Lu and the constrained distances for trees, and for the generic edit distance. The less-constrained distance is the only exception. In the diagram below, the trace mappings of $(b \to b', c \to c', d \to d')$ and $(b' \to b'', c' \to c'', d \to d'')$ meet the No-Twist condition. Nevertheless, $(b \to b'', c \to c'', d \to d'')$ does not, and hence is not in $T_{\lambda \gamma}^{d_i}$.

![Diagram](https://via.placeholder.com/150)

Based on this triplet of trees, we can determine a Gram matrix that is not positive definite for the less-constrained edit distance. For example, we first label all the vertices of $x$ and $z$ with “A” and all the vertices of $y$ with “B”. Secondly, we determine the cost function $\gamma$ such that all the operations costs 1 except that replacing a label with the same one costs 0. Then, for $\gamma^{\lambda \gamma} = 2$, the determinant of the resulting Gram matrix turns out negative. Furthermore, when all the vertices are labeled with the same single label, $d(x, y) = d(y, z) = 1$ and $d(x, z) = 4$ holds for the less-constrained distance, and hence, the triangle inequality does not hold.

### 4.5 Computational Complexity

The time complexity to compute $K_{\lambda \gamma d_i}^\lambda(x, y)$ relies on the edit distance definition used. For example, it is $O(|x||y|)$ for Levenshtein distance, $O(|x||y|)$ for Lu distance and the constrained tree edit distance, and $O(|x|^3)$ for Tai distance. The time complexity for the less-constrained tree edit distance falls between them and is $O(|x||y| \max((\deg x)^2, (\deg y)^2))$, where the degree $\deg x$ is the maximum of the number of the children of a single parent in $x$.

To compute them, we first convert $K_{\lambda \gamma d_i}^\lambda$ into $K_{\lambda \gamma d_i}^\lambda$, as shown in the proof to Lemma 3, and then compute $K_{\lambda \gamma d_i}^\lambda$ based on the dynamic programming methodology using recursive formulas derived in the same way as we derive the recursive formulas to compute the base edit distance. For these known cases, the space complexity is the same as the time complexity.

### 5. Empirical Results

The purpose of our experiments is to verify the feasibility of alignment kernels.

#### 5.1 Kernels and Datasets

As the alignment kernels to evaluate, we choose the ones that are based on Tai, Lu and the constrained tree edit distances (denoted by Tai, Lu and Constr. kernels). The align-
ment kernel based on the less-constrained distance is excluded, because it is not positive definite. To compare, we select two benchmark tree kernels presented in the literature, that is, the elastic tree kernel ([8]) and the parse tree kernel ([9]), denoted by Elastic and Parse kernels. The alignment kernels and the benchmark kernels can be calculated within the dynamic programming framework by taking advantage of the algorithms and the accelerating techniques introduced in [8]–[11]. Furthermore, the kernel values are used after being normalized, that is, the kernel values are converted into \( \frac{K(x,y)}{\sqrt{K(x,x)K(y,y)}} \) when input to SVM.

We use four datasets of trees. Three are retrieved from the KEGG/GLYCAN database ([12]), and contain glycan structures annotated relating to colon cancer, cystic fibrosis and leukemia cells. The remaining one, used in [13], consists of trees representing Web-page accesses by users, and the annotation is based on whether the user is from a .edu site or not. Table 1 describes the features of these datasets.

### 5.2 Training and Testing

For each dataset, we generate ten pairs of training and test data subsets by splitting it at random. For each pair, we train a C-SVM classifier with the training data subset and measure the AUC-ROC Curve scores by letting the trained classifier predicate class labels for the test data subset.

Also, when training the classifier, we simultaneously determine optimal values for the parameters of \( \lambda \) of \( K^d_{Aw} \) and \( C \) of C-SVM by grid search: We perform ten-fold cross validation on the training data subset changing the values for the parameters, and deploy the parameter value that exhibits the best AUC score. To make this optimization efficient to understand the property in time efficiency by deploying the accelerating technique introduced in [11] as well as by distributes the job of generating template Gram matrices over four processes (cores). We run the program with Intel

### 5.3 Results in Accuracy

Table 2 shows the AUC-ROC Curve measurements for each pair of a kernel and a dataset. The measurements are the averages across the ten pairs of training and test data subsets. Also, the values displayed in the parentheses are the ranks among the kernels, and are used in the statistical tests of Friedman and Shaffer static tests.

| Dataset   | \# Example | Av. Size | Av. Height |
|-----------|------------|----------|------------|
| Colon-cancer  | 134        | 8.4      | 5.6        |
| Cystic      | 160        | 8.3      | 5.0        |
| Leukemia    | 442        | 13.5     | 7.4        |
| Web-access  | 500        | 12.0     | 4.3        |

Table 2 shows us that \( T \) kernel always ranks top, and Const. and LU kernels follow in this order. To justify this observation, we perform Friedman test and Shaffer statistic test following the recommendations by Demšar ([14]) and García et al. ([15]): Demšar recommended use of non-parametric multiple comparison tests to compare multiple classifiers through experiments with multiple datasets, and García did Shaffer statistic test to compare all possible pairs of classifiers.

The null hypothesis of Friedman test is that there is no difference among the kernels, and the computed \( p \)-value is \( 6.14 \times 10^{-5} \). This value is very small, and we can reject the null hypothesis to conclude that there exists significant difference among the kernels. To know which pairs of kernels are actually different, we apply Shaffer static test. Table 3 shows the \( p \)-values computed for all the pairs. With the significance level \( \alpha = 0.1 \), we can reject the null hypotheses for the pair of \( T \) and Parse kernels and the pair of Const. and Parse kernels. Also, the \( p \)-value for the pair of \( T \) and Elastic kernels is relatively small.

### 5.4 Results in Time Efficiency

In the experiments stated above, generating template Gram matrices for \( T \) kernel took 4 seconds (Colon) to 17 minutes (Web-access), and LU and Const. kernels were approximately 2.5 times faster than \( T \) kernel. This observation is, however, insufficient to understand the property in time efficiency of the kernels correctly, and hence, we make additional experiments with randomly generated synthetic datasets that vary in the number and the size (the number of vertices) of the trees included. The program that we use for the experiments improves its time efficiency by deploying the accelerating technique introduced in [11] as well as by distributes the job of generating template Gram matrices over four processes (cores). We run the program with Intel

| Dataset   | \( T \) | LU | Const. | Elastic | Parse |
|-----------|--------|----|--------|--------|-------|
| Colon     | .953   | .930 | .953   | .930   | .874  |
|           | (1.5)  | (3.0) | (1.5)  | (4.0)  | (5.0) |
| Cystic    | .746   | .734 | .744   | .732   | .65   |
|           | (1.0)  | (3.0) | (2.0)  | (4.0)  | (5.0) |
| Leukemia  | .932   | .931 | .931   | .931   | .894  |
|           | (1.0)  | (3.0) | (3.0)  | (3.0)  | (5.0) |
| Web-access| .700   | .684 | .698   | .652   | .633  |
|           | (1.0)  | (3.0) | (2.0)  | (4.0)  | (5.0) |
| Average   | .833   | .82  | .831   | .811   | .763  |
|           | (1.125)| (3.0) | (2.125)| (3.75) | (5.0) |

Table 3 \( p \)-values of Shaffer statistic test.

|         | \( T \) | LU | Const. | Elastic | Parse |
|---------|--------|----|--------|--------|-------|
| \( T \) | .877   | 1.0| .113   | .005   |       |
| LU      | 1.0    | 1.0| .442   |        |       |
| Const.  | .584   | .061|       |        |       |
| Elastic | 1.0    |    |        |        |       |
| Parse   |       |    |        |        |       |
Xeon X5690 3.47GHz.

Figure 2 shows the time to compute a single kernel value (a polynomial of $\lambda$) changing the size $|x| = |y|$ of the input trees $(x, y)$ from 10 to 100. Each value displayed is an average across 55 computations. Since the time complexity to compute $\text{Ta}^i$ kernel is $O(|x|^3)$ (Section 4.5), its runtime increases rapidly as $|x|$ increases. On the other hand, $\text{Lu}$ and the $\text{Const.}$ kernels turn out as fast as each other and much faster than $\text{Ta}^i$ kernel.

Figure 3 shows the total time to compute a single template Gram matrix for datasets whose size varies from 10 to 100. The left chart presents the results for trees of size 50, while the right chart does for trees of size 100. In drawing the right chart, since the runtime records of $\text{Ta}^i$ kernel are large, (approximately 16 hours for 100 trees), we exclude it. Instead, the curve named $\text{Lu}$ (Naïve) shows the time necessary to compute $\text{Lu}$ kernel without the accelerating technique of [11]: Our program computes $\text{Lu}$ kernel faster by approximately 20% than the naive implementation.

5.5 Conclusions of the Experiments

The alignment kernels of $\text{Ta}^i$, $\text{Const.}$ and $\text{Lu}$ outperformed the benchmark kernels of $\text{Elastic}$ and $\text{Parse}$ in accuracy. In particular, the superiority of $\text{Ta}^i$ and $\text{Const.}$ kernels to $\text{Parse}$ kernel turns out statistically significant through Shaffer static test. With respect to time efficiency, $\text{Ta}^i$ kernel is significantly slower than the other kernels due to its time complexity of $O(|x|^3)$. On the other hand, $\text{Const.}$ and $\text{Lu}$ kernels are approximately 50% slower than $\text{Elastic}$ kernel, and are sufficiently fast in practice. Overall, we recommend use of $\text{Const.}$ kernel which has showed a good balance between accuracy and time efficiency.

6. Conclusions

We have introduced alignment kernels that can be defined based on edit distances for general graphs and whose positive definiteness can be easily investigated. For this purpose, we have generalized the notion of alignments, which were defined only for particular classes of edit distances in the literature, and then, have defined alignment kernels based on the generalized notion of alignments and the soft minimum approximation technique.

Also, we took tree kernels as an example, and investigated three different types of alignment kernels, that is, those derived from $\text{Ta}^i$, $\text{Lu}$ and the constrained edit distances through experiments to compare them with benchmark tree kernels, that is, the elastic kernel and the parse tree kernel. The results have shown that the alignment kernels outperformed the benchmark kernels in accuracy, while they exhibited lower time efficiency. Although $\text{Ta}^i$ kernel might be too slow to apply to large datasets, the constrained-distance-based kernel is reasonably fast, and provides a good balance of accuracy and time efficiency.

References

[1] C. Berg, J.P.R. Christensen, and R. Ressel, Harmonic Analysis on semigroups. Theory of positive definite and related functions, Springer, 1984.
[2] I.J. Schoenberg, “Metric spaces and positive definite functions,” Trans. Amer. Math. Soc., vol.44, pp.522–536, 1938.
[3] C. Cortes, P. Haffner, and M. Mohri, “Rational kernels: Theory and algorithms,” J. of Machine Learning Research, vol.1, pp.1–50, 2004.
[4] K. Riesen and H. Bunke, “Graph classification by means of Lipschitz embedding,” IEEE Trans. Syst., Man Cybern. B: Cybern., vol.39, no.6, pp.1472–1483, 2009.
[5] M. Neuhaus and H. Bunke, “Edit distance-based kernel functions for structural pattern classification,” Pattern Recognit., vol.10, pp.1852–1863, 2006.
[6] K. Shin and T. Kuboyama, “A generalization of Haussler’s convolution kernel - mapping kernel,” 25th International Conference on Machine Learning (ICML 2008), pp.944–951, 2008.
[7] K. Shin, “Edit distance and alignment kernels,” International Conference on Artificial Intelligence, Soft Computing, 2013.
[8] H. Kashima and T. Koyanagi, “Kernels for semi-structured data,” 9th International Conference on Machine Learning (ICML 2002), pp.291–298, 2002.
[9] M. Collins and N. Duffy, “Convolution kernels for natural language,” Advances in Neural Information Processing Systems 14 (NIPS 2001), pp.625–632, MIT Press, 2001.
[10] T. Kuboyama, K. Shin, and H. Kashima, “Flexible tree kernels based on counting the number of tree mappings,” Proc. of Machine Learning with Graphs, 2006.
[11] K. Shin, M. Cuturi, and T. Kuboyama, “Mapping kernels for trees,” 28th International Conference on Machine Learning (ICML 2011), pp.961–968, 2011.
[12] K. Hashimoto, S. Goto, S. Kawano, K.F. Aoki-Kinoshita, and N. Ueda, “Kegg as a glycemic informatics resource,” Glycobiology, vol.16, pp.63R–70R, 2006.
Appendix A: Transformation to traditional Alignments

As seen in 3.3, for Levenshtein distance and the less-constrained distance, the alignment graphs are different from the alignments according to the legacy definitions. In this subsection, we determine four primitive graph transforming operations by which we can transform alignment graphs into traditional alignments (Table A·1).

Given an alignment graph \( \iota: \bar{x} \rightarrow \bar{y} \in \mathcal{A}_{x,y} \), we first draw its dual-label representation \( g \). The graph \( g \) is canonically homeomorphic to \( \bar{x} \) and \( \bar{y} \), and its vertices and edges are labeled with \((\ell_{V}\bar{x}(v), \ell_{V}\bar{y}(\iota V(v)))\) and \((\ell_{E}\bar{y}(e), \ell_{E}\bar{y}(\iota E(e)))\). For example, we can transform Diagram (3) into Diagram (4) by applying S2 three times and EU twice. In fact, the following depicts the result of having applied S2 twice.

![Diagram](image)

One thing to note here is that the way of transformation is not unique. In fact, the positions of \( a \bullet \) and \( i \bullet \) in Diagram (4) can be exchanged.

An alignment tree for the less-constrained distance is also derived from the alignment graph of our definition applying the same transformation rules. In fact, the top-right diagram of Fig. A·1 is the dual-label representation of Diagram (5) and the left-hand path shows how this alignment graph can be transformed to Diagram (3). The right-hand path shows another result of applying the same rules, and the result is not a tree.

When a trace mapping does not meet the No-Twist condition, the result of applying the transformation rules cannot be a tree. We obtain the following by applying the rules to Diagram (6).

![Diagram](image)

We can formulate the discussion above as Theorem 7.

**Theorem 7:** The alignment graph associated with a trace mapping for Taü distance can be transformed to a tree by means of the rules determined in Table A·1, if, and only if, the trace mapping meets the No-Twist condition.

Appendix B: On Negative Definiteness of the Edit Distances that We Surveyed

While Cortes et al. have proven that Levenshtein distance for strings is not a negative definite kernel ([3]), the same
result holds true for Tâ, Lu, the constrained and the less-constrained edit distances for trees and for the generic graph edit distance. This can be proven based on the following two facts: Strings can be naturally viewed as trees and graphs; An alignment graph between strings yields an alignment graph when we view the strings as trees and graphs, and the converse is also true.

In fact, we can identify a string with a linear tree such that one end of the string is the root and the other is the unique leaf and as a graph that includes no edge. It is apparent from the definition that the set of alignment graphs between two strings is identical to the set of alignment graphs between trees or graphs under this identification. Finally, we can apply Theorem 3 to see that Levenshtein distance between the strings is identical to their distances as trees and graphs.

Kilho Shin has got his MS in mathematics and Ph.D. in engineering from University of Tokyo. After working for Corporate Research Center of Fuji Xerox and Research Center of Advanced Science and Technologies of University of Tokyo, he is currently working at Graduate School of Applied Informatics of University of Hyogo. He is also a faculty of Carnegie Mellon University in US. The scope of his research activity includes public key cryptography, privacy protection, kernel design for discrete structures and feature selection.