Pattern Recognition 44 (2011) 1172–1182

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

A sum-over-paths extension of edit distances accounting for all sequence alignments

Silvia García-Díez [a,*], François Fouss [b], Masashi Shimbo [c], Marco Saerens [a]

[a] Université de Louvain, ISYS, LSM, Louvain-la-Neuve, Belgium
[b] Facultés Universitaires Catholiques de Mons, Management Science Department, LSM, Belgium
[c] Graduate School of Information Science, Nara Institute of Science and Technology, Japan

ARTICLE INFO

Article history:
Received 7 October 2009
Received in revised form
29 November 2010
Accepted 30 November 2010
Available online 3 December 2010

Keywords:
Edit distance
Longest common subsequence
Sequence comparison
Approximate string matching
Dynamic programming
Viterbi algorithm
Shortest path

ABSTRACT

This paper introduces a simple Sum-over-Paths (SoP) formulation of string edit distances accounting for all possible alignments between two sequences, and extends related previous work from bioinformatics to the case of graphs with cycles. Each alignment $\wp$, with a total cost $C(\wp)$, is assigned a probability of occurrence $P(\wp)=\exp[-\theta C(\wp)]/\mathcal{Z}$ where $\mathcal{Z}$ is a normalization factor. Therefore, good alignments (having a low cost) are favored over bad alignments (having a high cost). The expected cost $\sum_{\wp\in\mathcal{P}} C(\wp)\exp[-\theta C(\wp)]/\mathcal{Z}$ computed over all possible alignments $\wp\in\mathcal{P}$ defines the SoP edit distance. When $\theta\to\infty$, only the best alignments matter and the measure reduces to the standard edit distance. The rationale behind this definition is the following: for some applications, two sequences sharing many good alignments should be considered as more similar than two sequences having only one single good, optimal, alignment in common. In other words, sub-optimal alignments could also be taken into account. Forward/backward recurrences allowing to efficiently compute the expected cost are developed. Virtually any Viterbi-like sequence comparison algorithm computed on a lattice can be generalized in the same way; for instance, a SoP longest common subsequence is also developed. Pattern classification tasks performed on five data sets show that the new measures usually outperform the standard ones and, in any case, never perform significantly worse, at the expense of tuning the parameter θ.

1. Introduction

1.1. General introduction

Approximate string matching (ASM) algorithms find applications in various areas, such as pattern recognition [57], computer science [12,54], bioinformatics [17,29,51], and computational linguistics [27]. Numerous extensions have been proposed, allowing either to improve the performance of the basic techniques or to extend their applicability to new problems for which the basic techniques are not suited.

The aim of this work is to extend, in a straightforward way, the basic ASM algorithms in the following way. Most of the ASM algorithms return the score provided by the best, optimal, alignment between two sequences. By optimal, we mean the alignment corresponding to the least total cost, which can be computed by the well-known Viterbi algorithm [20]. Now, instead of only relying on the best alignments, we propose to average the total cost over all the possible alignments. Indeed, as noticed in [17], relevant information is also contained in the sub-optimal paths. For some applications, two sequences sharing many good alignments should be considered as more similar than two sequences having only one single good, optimal, alignment in common. To this end, we adopt a Sum-over-Paths (SoP) formalism, considering that each alignment corresponds to a path on the dynamic programming lattice. This formalism is directly inspired by [49], which was itself inspired by the work of Akamatsu in transportation sciences [3]. The same idea has already appeared in bioinformatics for acyclic lattices [40,70] (see Section 1.2). The SoP generalizes this idea to arbitrary graphs.

The basic idea is to consider the set of all possible alignments and to weight their contribution to the total cost according to their respective quality. Each alignment is weighted by a probability mass penalizing bad alignments, so that the contribution of better alignments is more important. Concretely, the alignment choice is *weighted* by imposing a pre-defined entropy level. The probability distribution on the set of alignments is defined as the one minimizing the total expected cost for the predefined entropy. It turns out that this optimal probability distribution is a Boltzmann distribution on the set of alignments (see Section 2.3 or [49]), depending on a parameter θ—the inverse temperature which is inversely related to the entropy – controlling the trade-off between the contribution of good and bad alignments. The SoP edit distance

* Corresponding author.
E-mail addresses: silvia.garciadiez@uclouvain.be (S. García-Díez), francois.fouss@fucam.ac.be (F. Fouss), shimbo@is.naist.jp (M. Shimbo), marco.saerens@uclouvain.be (M. Saerens).

0031-3203/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2010.11.020

is then defined as the total expected cost computed over all possible alignments, the expectation being taken according to the Boltzmann distribution.

It is shown that this total expected cost can be easily computed within this framework. The resulting algorithm is similar to the forward/backward algorithm used in training hidden Markov models: the recurrence relations allowing to compute the different quantities in an efficient way are readily derived. In particular, our algorithm reduces to the standard forward/backward algorithm when the θ parameter is equal to 1. On the other hand, when $\theta \to \infty$, it corresponds to the Viterbi algorithm and thus reduces to the basic algorithm computing only the best alignments. Two standard edit distance algorithms are extended within this framework: the Levenshtein edit distance (LED) and the longest common subsequence (LCS); they will respectively be called the SoP edit distance (SoP ED) and the SoP common subsequence similarity (SoP CS).

1.2. Related work

As discussed in [49], the idea of randomizing the policy was introduced by [1,2] in the context of reinforcement learning and was inspired by the entropy rate of an ergodic Markov chain defined in information theory (see, e.g., [13]). In this previous work, the sum of the local entropies on all nodes was fixed and the optimal policy for this fixed level of entropy was computed through a value-iteration-like algorithm.

In [49], instead of fixing the local entropy defined at each node as in [1,2], the global entropy spread over the whole network is fixed. While this difference seems a priori insignificant, it appears that constraining the global spread entropy of the network is more natural and easier to compute. Clearly, the nodes that need a large spread are difficult to determine in advance, and the model has to distribute the entropy by optimizing it globally all over the network. It was shown in [49] that the optimal randomized policy can be found by solving a simple system of linear equations. In the same paper, the authors showed that when the graph is acyclic—and therefore a lattice—the expected cost as well as the policy can be computed efficiently from two simple recurrence relations. This fact is exploited in this paper in order to define a randomized edit distance.

Apart from this previous work [1,2,49], and some others in game theory (see for instance [42]) or Markov games [34], very few optimal randomized strategies have been exploited in the context of shortest-path problems. There is, however, one exception: Akamatsu's model [3], who designed a randomized policy for routing traffic in transportation networks. In transportation science, randomized strategies are called *stochastic traffic assignments* and, within this context, Akamatsu's model is the model of reference. This work, as well as [49], are inspired by Akamatsu's model.

Let us also mention some papers that are related to path randomization, and therefore to the present work. The entropy of the paths (or trajectories) connecting an initial and an absorbing destination node of an absorbing Markov chain was studied in [18]. In this paper, the authors provide formulae allowing to compute the entropy needed to reach the destination node. In [56] a one-parameter family of algorithms that recover both the procedure for finding shortest paths as well as the iterative algorithm for computing the average first-passage time in a Markov chain is introduced. However, having a heuristic foundation, it is not based on the optimization of a well-defined cost function. In another context, Todorov [60] studied a family of Markov decision problems that are linearly solvable, that is, for which a solution can be computed by solving a matrix eigenvector problem. In order to make this possible, Todorov assumes a special form of control for the transition probabilities, which recasts the problem of finding the policy into an eigenvector problem. In [9] a Markov chain that has the fastest mixing properties is designed, while [55] discuss its continuous-time counterpart. In a completely different framework, uninformed random walks, based on maximizing the long-term entropy [15,61] have recently been proposed as an alternative to the standard PageRank algorithm. Finally, notice that some authors tackled the problem of designing ergodic (non-absorbing) Markov or semi-Markov chains in a maximum-entropy framework (see for instance [21,22] and the references therein).

Within the context of computing edit distances between sequences, there have been successful attempts to account for all possible alignments (see, e.g., [10,17,28,31,35,48,52,68], among others). The sequence comparison model introduced in [17] is quite popular in bioinformatics. It is based on a hidden Markov model (HMM) of sequence pairs generation, called the pair HMM. The model is trained by maximum likelihood on a sequences sample and, once is trained, it provides the likelihood of observing any two sequences, each alignment being weighted by its probability. The SoP edit distance introduced in this paper shares therefore some similarities with the pair HMM, but is much simpler since it does not require any transition-probability estimation, i.e., it is model-free, although by modifying the edition costs or the similarity measure we could adapt it to any model or specific task. It is, however, also possible to fix a priori the transition probabilities of the pair HMM. In that case, the SoP edit distance is equivalent to the pair HMM with a suitable choice of the editing costs, and parameter θ equal to 1. However, two important differences between the proposed SoP techniques and pair HMMs is that (i) it weights the contribution of the different alignments, according to their respective total costs, and (ii) it depends on a parameter allowing to regulate the degree of exploration. It therefore encompasses both the Viterbi and the Baum–Welch algorithms as special cases. It will be observed in the experimental section that the best performance obtained by the SoP techniques is often achieved for θ parameter values greater than 1. Finally, notice that a valid kernel derived from a pair HMM was proposed in [26,68].

The string kernels introduced in [31,33,35,48,50,52] compute a score depending on all the possible common subsequences of the two compared sequences. The main idea behind these sequence-comparison techniques is to reward common subsequences, contiguous or not, depending on the technique. However, when the size of the alphabet is very low, there is a huge quantity of such short common subsequences and very few long common subsequences, so that the obtained score does not reflect accurately the proper similarity between the two sequences. Indeed, a much larger weight should be put on long common subsequences; this is exactly what the SoP edit distance does. We therefore expect string kernels to perform well when the alphabet is quite large (for instance in the context of text mining—in this case—there are very few long common subsequences; see for example [11]), and worse in the case of reduced alphabets—this is indeed observed in our experiments.

Yet another approach computing an edit score along all possible alignments is the stochastic edit distance, and related methods [6]. The underlying model is based on a noisy channel. An input string $\mathbf{x}$—the reference string or code—is distorted by a noisy channel, therefore producing an output string $\mathbf{y}$ that is a noisy transform of $\mathbf{x}$. The noisy channel is often modeled as a Markov model or a transducer [6,41]. For these models, the probability of generating string $\mathbf{y}$ from $\mathbf{x}$, $P(\mathbf{y}|\mathbf{x})$, can be computed thanks to a forward recursion formula involving all possible ways of generating $\mathbf{y}$, and therefore all possible paths through the lattice. The scores $-\log P(\mathbf{y}|\mathbf{x})$ or $-\log P(\mathbf{x}|\mathbf{y})$ can be considered as dissimilarity measures between $\mathbf{x}$ and $\mathbf{y}$. This stochastic edit distance model is refined in [47,4,41], where the local distances between the symbols are estimated from sample data. A paragraph explaining the relationships between stochastic edit distances and the sum-over-paths approach appears at the end of Section 2.4. However, these

stochastic edit distances do not have a parameter biasing the measure towards minimum cost solutions.

Path randomization applications are also found within the bioinformatics literature. A stochastic alignment procedure is presented in [58], where the authors introduce an evolutionary model based on transition probabilities which allow not only to estimate the optimal alignment via the maximum likelihood, but also to estimate the set of evolutionary parameters given two aligned sequences. This work is further extended by [59] by adding the possibility of multiple-base insertion and deletion as well as heterogeneous evolutionary rates. This evolutionary model is also extended by [53] by generalizing the pairwise recursions to an r-sequence star-shaped tree. In [70], the authors propose a method which extends the typical dynamic programming method by allowing uncertainty and, thus, weighting the fluctuating sub-optimal alignments. This probability distribution has its basis on the partition function, which is computed via a recurrence formula. The average cost is also computed, though is not used as similarity measure between two sequences. Furthermore, this model is restricted to lattices and there are no methodic experimental results, although simulation experiences are suggested. In [38] a formulation for the alignments with highest probability, according to a Boltzmann distribution based on the partition function and a temperature parameter, is provided. The author defines a pairwise similarity measure consisting on a log-odds matrix of amino acid mutations, which is then used to compute the statistical weight of an alignment. Moreover, a formulation of pairwise probabilities in terms of forward and backward variables is presented. Eventually, an alignment consisting of the most probable pairwise correspondences is provided. In [25], the authors introduce a similarity detection method based on statistical physics. The idea behind this paper is that the large-scale statistics of the fluctuating paths can be derived from the partition function in a path integral representation. Other similar probability alignments are also presented in [30], where the finite-temperature alignments are introduced. These stochastic alignments are derived from a thermodynamic partition function and they can be used for assessing the relevance of sub-optimal paths. The authors define the weight of each alignment proportionally to the exponential of a defined similarity measure which depends on the number of matches, mismatches and gaps. Eventually, the probability of a pairwise element is computed as the sum of the probabilities over all paths that contain the pair. This partition function formalism is also computed in [40] via a dynamic programming algorithm, where the authors introduce a probabilistic approach to calculate the alignment score based on a Boltzmann distribution.

However, and to the best of our knowledge, there is no work that uses the average cost of the fluctuating sub-optimal paths as a similarity measure between two sequences, yet many of them agree that the optimal alignment may not be the best one, and that other sub-optimal paths are also relevant. The present work introduces a SoP formalism which allows to compute relevant quantities on a graph, such as the expected cost or the expected number of passages through a node. A probability distribution is assigned on the set of paths and the quantities of interest are computed with respect to this distribution. The derivation of these quantities differs from [36,49,69] in that it is more intuitive, in our opinion, as it directly derives the main quantities without having to compute the derivatives of a partition function. Within this formalism, extensions of the edit distance and the longest common subsequence are developed, taking all the alignments into account.

It must also be stressed that these previous works focus only on acyclic lattices, while the presented approach is generalized to arbitrary graphs (including loops). Furthermore, although the use of the partition function formalism proves to be useful in this context, any of the previous work derives it from an optimization problem (as is the case of the sum-over-paths edit distances). Eventually, previous work is mostly focused on biologically relevant similarity or cost functions, while generic edit distance costs are applied here, leading to an application-independent method which proves to be useful in OCR tasks.

Finally, let us also mention that some authors [16,41,45,46] propose to tune the edit costs from a training set, which results in better performances at the cost of increasing computational complexity. Moreover, an edit graph (lattice) for a pair of sequences can be thought of as a strong product graph of two chain graphs each representing one of the two given sequences, the SoP approach is also related to graph kernels working on product graphs [63,64]. For instance, Vishwanathan et al.'s framework [63] of graph kernels viewed as path counting in product graphs could probably be adapted to strong product graphs and thus applied to sequence comparison.

2. The sum-over-paths edit distance and common subsequence

2.1. The standard edit distance

Let us start from the well-known dynamic programming framework for computing the standard edit distance [23,39,54,66]. The Levenshtein edit distance (LED) between two sequences $\mathbf{s}_1$, $\mathbf{s}_2$ of length n_1, n_2 is defined as the minimum number of symbol editing operations (typically substitutions, deletions, insertions) needed to transform the first sequence into the second one. The dynamic programming computation occurs on a lattice where each node k corresponds to a state of the editing procedure (the i first symbols of $\mathbf{s}_1$ have been transformed into the j first symbols of $\mathbf{s}_2$). A link $k \to k'$ between two nodes defines a possible editing operation that transforms state k into state k', and an immediate cost $c_{kk'}$ is associated to each editing operation[1] causing a transition from node k to k'. If there is no direct link between k and k', the immediate cost is assumed to be infinite, $c_{kk'} = \infty$. Let us consider that the nodes have been reordered in such a way that node 1 is the starting node and node n is the ending node; moreover, a topological ordering has been performed. The resulting directed graph contains $n = (n_1+1) \times (n_2+1)$ nodes and is acyclic; it corresponds to an acyclic lattice. It is assumed that this lattice has been pre-computed and that, for each node k, the sets Pred(k) of predecessor nodes and successor nodes Succ(k) are available.

In this case, the recurrence equations for computing the standard **Levenshtein edit distance** (LED) are

$$\begin{cases} d_1 = 0 \\ d_{k'} = \min\limits_{k \in \mathrm{Pred}(k')} (c_{kk'} + d_k) \end{cases} \quad (1)$$

where Pred(k') is defined as the set of predecessors of node k'. The edit distance is then defined as d_n. This standard formulation only considers the optimal alignment between the two sequences, and can be viewed as a Viterbi algorithm [20].

2.2. The standard longest common subsequence

The length of the longest subsequence common to two strings can be regarded as a measure of similarity between the two. The longest common subsequence (LCS) algorithm [12,54] computes this similarity (together with the longest common subsequence itself, if desired) through a method similar to the edit distance computation. In Section 2.3, we develop a SoP-based extension of

[1] Usually, the same cost is assigned to insertion and deletion operations and the substitution costs are assumed to be symmetric—otherwise the resulting measure would not be symmetric.

the LCS algorithm that takes all common subsequences into account, while still favoring longer sequences over shorter ones. The trade-off between both types of sequences is controlled by the parameter θ.

2.3. A sum-over-paths extension of the edit distance

Let us now turn to our sum-over-paths (SoP) formalism, computing the distance along all the alignments between the two sequences. It is a direct application of the randomized shortest-path formalism developed in [49], inspired by the work of Akamatsu [3] appearing in transportation sciences. The derivation provided in this paper is more intuitive than the one appearing in [49].

We first consider the set of all possible paths $\mathcal{P}_{1n}$ between the starting node 1 and the ending node n on the dynamic programming lattice. Each of these paths $\wp \in \mathcal{P}_{1n}$ corresponds to a possible alignment between sequence $\mathbf{s}_1$ and sequence $\mathbf{s}_2$. The probability of occurrence of each alignment is assumed to follow a **Boltzmann distribution** as shown in Eq. (2)

$$P(\wp) = \frac{\exp[-\theta C(\wp)]}{\mathcal{Z}} \tag{2}$$

$$\mathcal{Z} = \sum_{\wp \in \mathcal{P}_{1n}} \exp[-\theta C(\wp)] \tag{3}$$

where $C(\wp)$ is the total cost associated to the path or alignment $\wp$, that is, the sum of the individual costs $c_{kk'}$ occurring on that path $\wp$. $\mathcal{Z}$ (Eq. (3)) is the normalization factor (called the partition function [36]). This partition function has already been used for sequence alignment with uncertainty (see i.e., [30,38,70]), for similarity detection as in [25], or in alignment scoring as in [40].

In fact, the probability distribution defined in Equation (2) minimizes the expected cost subject to a Shannon entropy constraint,

$$\left| \begin{array}{ll} \underset{P(\wp)}{\text{minimize}} & \sum_{\wp \in \mathcal{P}_{1n}} P(\wp)C(\wp) \\ \text{subject to} & -\sum_{\wp \in \mathcal{P}_{1n}} P(\wp)\ln P(\wp) = H_0 \end{array} \right. \tag{4}$$

where H_0 is a predefined entropy [36] and is inversely related to θ (the lower θ, the larger the entropy), the inverse temperature parameter. In other words, it guarantees a minimum expected cost for reaching node n from node 1 while fixing the level of entropy (exploration) spread in the lattice.

It is clear that this definition of the probability distribution on the set of paths favors good alignments (with a low cost $C(\wp)$) over bad ones (with a large cost $C(\wp)$). The parameter θ regulates the sharpness of the distribution: when $\theta \to \infty$, only the best alignments matter while when $\theta \to 0$, all alignments have almost the same probability mass. In other words, the larger the value of parameter θ, the less the impact of sub-optimal paths.

Now, the total expected cost for reaching node n from node 1, defining the **SoP edit distance** d_{SoP} between the two sequences, is

$$d_{\text{SoP}} = \mathbb{E}[C] = \sum_{\wp \in \mathcal{P}_{1n}} P(\wp)C(\wp) = \sum_{\wp \in \mathcal{P}_{1n}} C(\wp)\frac{\exp[-\theta C(\wp)]}{\mathcal{Z}} \tag{5}$$

On the other hand, the expected number of passages through the node k and through the link $k \to k'$ can be expressed as

$$n_k = \sum_{\wp \in \mathcal{P}_{1n}} \delta(\wp; k)\frac{\exp[-\theta C(\wp)]}{\mathcal{Z}} \tag{6}$$

$$n_{kk'} = \sum_{\wp \in \mathcal{P}_{1n}} \delta(\wp; k,k')\frac{\exp[-\theta C(\wp)]}{\mathcal{Z}} \tag{7}$$

where $\delta(\wp; k)$ and $\delta(\wp; k,k')$ are indicator functions counting the number of times node k (or link $k \to k'$) is traversed by path $\wp$. If the graph is acyclic (for instance an acyclic lattice), so that a path can only visit a link once, it returns 1 if, respectively, the node k or the link $k \to k'$ is part of path $\wp$, and 0 otherwise. Thus, the total cost occurred on path $\wp$ can be computed from the individual costs $c_{kk'}$ by

$$C(\wp) = \sum_{k=1}^{n}\sum_{k'=1}^{n} \delta(\wp; k,k')c_{kk'} \tag{8}$$

Therefore, from Eq. (5), the total expected cost can be rewritten as

$$d_{\text{SoP}} = \sum_{\wp \in \mathcal{P}_{1n}} C(\wp)\frac{\exp[-\theta C(\wp)]}{\mathcal{Z}} = \sum_{\wp \in \mathcal{P}_{1n}} \left(\sum_{k=1}^{n}\sum_{k'=1}^{n} \delta(\wp; k,k')c_{kk'}\right)\frac{\exp[-\theta C(\wp)]}{\mathcal{Z}} \tag{9}$$

$$= \sum_{k=1}^{n}\sum_{k'=1}^{n} c_{kk'}\left(\sum_{\wp \in \mathcal{P}_{1n}} \delta(\wp; k,k')\frac{\exp[-\theta C(\wp)]}{\mathcal{Z}}\right) = \sum_{k=1}^{n}\sum_{k'=1}^{n} c_{kk'}n_{kk'} \tag{10}$$

where we used the definition provided by Eq. (7). Thus, the SoP edit distance is just the sum of the individual costs multiplied by the expected number of passages through the corresponding links. By generalizing with the same argument, the expectation of any quantity $v_{kk'}$ (instead of the costs $c_{kk'}$) defined on the links is

$$\mathbb{E}[V] = \sum_{\wp \in \mathcal{P}_{1n}} P(\wp)V(\wp) = \sum_{k=1}^{n}\sum_{k'=1}^{n} v_{kk'}n_{kk'} \tag{11}$$

where $V(\wp)$ is the sum of the $v_{kk'}$ along the path $\wp$.

Let us now show that, as in hidden Markov models, the average number of passages through each node can easily be expressed in terms of forward and backward variables. Notice first that, since only the paths passing through node k contribute to the sum in Eq. (6), they can be split into two sub-paths (see Fig. 1): $\wp_{1k} \in \mathcal{P}_{1k}$ and $\wp_{kn} \in \mathcal{P}_{kn}$. These two sub-paths can be chosen independently since their composition is a valid path, where $\wp_{1k}\wp_{kn} \in \mathcal{P}_{1n}$ is the concatenation of the two paths.

Now, since $C(\wp) = C(\wp_{1k}) + C(\wp_{kn})$ for any $\wp = \wp_{1k}\wp_{kn}$, we easily obtain

$$n_k = \sum_{\wp \in \mathcal{P}_{1n}} \delta(\wp; k)\frac{\exp[-\theta C(\wp)]}{\mathcal{Z}} \tag{12}$$

$$= \frac{1}{\mathcal{Z}}\sum_{\substack{\wp_{1k} \in \mathcal{P}_{1k} \\ \wp_{kn} \in \mathcal{P}_{kn}}} \exp[-\theta C(\wp_{1k})]\exp[-\theta C(\wp_{kn})] \tag{13}$$

$$= \frac{1}{\mathcal{Z}}\sum_{\wp_{1k} \in \mathcal{P}_{1k}}\sum_{\wp_{kn} \in \mathcal{P}_{kn}} \exp[-\theta C(\wp_{1k})]\exp[-\theta C(\wp_{kn})] \tag{14}$$

$$= \frac{1}{\mathcal{Z}}\left(\sum_{\wp_{1k} \in \mathcal{P}_{1k}} \exp[-\theta C(\wp_{1k})]\right)\left(\sum_{\wp_{kn} \in \mathcal{P}_{kn}} \exp[-\theta C(\wp_{kn})]\right) = \frac{z_{1k}z_{kn}}{\mathcal{Z}} = \frac{z_{1k}z_{kn}}{z_{1n}} \tag{15}$$

where we defined the *forward variable* z_{1k} and the *backward variable* z_{kn} as

$$z_{1k} = \sum_{\wp \in \mathcal{P}_{1k}} \exp[-\theta C(\wp)] \tag{16}$$

$$z_{kn} = \sum_{\wp \in \mathcal{P}_{kn}} \exp[-\theta C(\wp)] \tag{17}$$

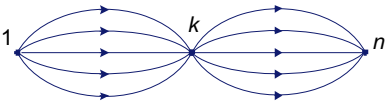

Fig. 1. The paths $\mathcal{P}_{1n}$ from 1 to n traversing node k can be decomposed into two sub-paths in $\mathcal{P}_{1k}$ and in $\mathcal{P}_{kn}$.

and it is clear from Eq. (3) that $\mathcal{Z}=z_{1n}$. Recurrence relations for computing the forward/backward variables will be derived in the next subsection (Subsection 2.4).

Interesting enough, Eq. (15) still holds in the case of arbitrary weighted directed graphs, and not only for acyclic lattices. In that situation, paths might contain loops and the decomposition $\wp=\wp_{1k}\wp_{kn}$ is no more unique: a single path $\wp$ can be decomposed in $\wp_{1k}\wp_{kn}$ in as many ways as the number of times $\wp$ passes through node k. Therefore, each path $\wp\in\mathcal{P}_{1n}$ is counted several times in the sum of Eq. (13): it appears as many times as the path $\wp$ can be decomposed in $\wp=\wp_{1k}\wp_{kn}$. But this quantity corresponds to the number of times node k appears in path $\wp$ which, in turn, is exactly equal to $\delta(\wp;k)$. Therefore, the passage from Eq. (12) to Eq. (13) remains valid for general graphs containing cycles.

The same reasoning holds for the average number of passages through each link. Indeed, only the paths passing through link $k\to k'$ contribute to the sum in Eq. (7). They can therefore be split into three sub-paths (see Fig. 2): $\wp_{1k}\in\mathcal{P}_{1k}$, $\wp_{kk'}\in\mathcal{P}_{kk'}$ and $\wp_{k'n}\in\mathcal{P}_{k'n}$. As before, the three sub-paths can be chosen independently since their composition is a valid path, $\wp_{1k}\wp_{kk'}\wp_{k'n}\in\mathcal{P}_{1n}$. Since $C(\wp)=C(\wp_{1k})+C(\wp_{kk'})+C(\wp_{k'n})$, we obtain

$$\begin{aligned}
n_{kk'} &= \sum_{\wp\in\mathcal{P}_{1n}}\delta(\wp;k,k')\frac{\exp[-\theta C(\wp)]}{\mathcal{Z}}\\
&= \frac{1}{\mathcal{Z}}\sum_{\substack{\wp_{1k}\in\mathcal{P}_{1k}\\ \wp_{kk'}\in\mathcal{P}_{kk'}\\ \wp_{k'n}\in\mathcal{P}_{k'n}}}\exp[-\theta C(\wp_{1k})]\exp[-\theta C(\wp_{kk'})]\exp[-\theta C(\wp_{k'n})]\\
&= \frac{1}{\mathcal{Z}}\sum_{\wp_{1k}\in\mathcal{P}_{1k}}\sum_{\wp_{kk'}\in\mathcal{P}_{kk'}}\sum_{\wp_{k'n}\in\mathcal{P}_{k'n}}\exp[-\theta C(\wp_{1k})]\exp[-\theta C(\wp_{kk'})]\exp[-\theta C(\wp_{k'n})]\\
&= \frac{1}{\mathcal{Z}}\left(\sum_{\wp_{1k}\in\mathcal{P}_{1k}}\exp[-\theta C(\wp_{1k})]\right)\left(\sum_{\wp_{kk'}\in\mathcal{P}_{kk'}}\exp[-\theta C(\wp_{kk'})]\right)\\
&\quad\times\left(\sum_{\wp_{k'n}\in\mathcal{P}_{k'n}}\exp[-\theta C(\wp_{k'n})]\right)=\frac{z_{1k}z_{kk'}z_{k'n}}{\mathcal{Z}}=\frac{z_{1k}\exp[-\theta c_{kk'}]z_{k'n}}{z_{1n}}
\end{aligned}\tag{18}$$

Once more, by the same argument, it can be shown that Eq. (18) still holds for graphs containing cycles. Finally, Eq. (10) can be rewritten as

$$\begin{aligned}
d_{\mathrm{SoP}} &= \frac{1}{z_{1n}}\sum_{k=1}^{n}\sum_{k'=1}^{n}z_{1k}c_{kk'}\exp[-\theta c_{kk'}]z_{k'n}\\
&= \frac{1}{z_{1n}}\sum_{k=1}^{n}\sum_{k'\in\mathrm{Succ}(k)}z_{1k}c_{kk'}\exp[-\theta c_{kk'}]z_{k'n}
\end{aligned}\tag{19}$$

which allows computing the SoP edit distance in terms of the immediate costs and the forward/backward variables.

The time and space complexity of this algorithm is $O(n_1\ n_2)$, where n_1 and n_2 are the length of the two input sequences; i.e., it has the same time complexity as the standard, non-optimized, edit distance computation. However, the space complexity of the edit distance is $O(\min(n_1,\ n_2))$ (by divide and conquer). The space complexity of the SoP distances can only be improved by increasing the computation time (see Section 3.4 for further details).

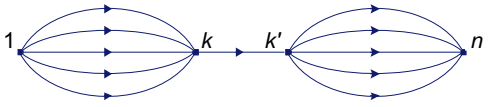

Fig. 2. The paths $\mathcal{P}_{1n}$ from 1 to n traversing link $k\to k'$ can be decomposed into three sub-paths respectively in $\mathcal{P}_{1k}$, in $\mathcal{P}_{kk'}$, and in $\mathcal{P}_{k'n}$.

2.4. Recurrence relations for computing the forward/backward variables

Let us now derive recurrence relations allowing us to compute these forward/backward variables efficiently. We first investigate the forward variables. If node $k\in\mathrm{Pred}(k')$ is a predecessor of node k' (see Fig. 3(a)), the paths $\mathcal{P}_{1k'}$ can be decomposed into $\mathcal{P}_{1k}$ and $\mathcal{P}_{kk'}$:

$$\begin{aligned}
z_{1k'} &= \sum_{\wp_{1k'}\in\mathcal{P}_{1k'}}\exp[-\theta C(\wp_{1k'})]=\sum_{k\in\mathrm{Pred}(k')}\sum_{\wp_{1k}\in\mathcal{P}_{1k}}\exp[-\theta(C(\wp_{1k})+C(\wp_{kk'}))]\\
&= \sum_{k\in\mathrm{Pred}(k')}\sum_{\wp_{1k}\in\mathcal{P}_{1k}}\exp[-\theta C(\wp_{1k})]\exp[-\theta C(\wp_{kk'})]\\
&= \sum_{k\in\mathrm{Pred}(k')}\exp[-\theta C(\wp_{kk'})]z_{1k}=\sum_{k\in\mathrm{Pred}(k')}\exp[-\theta c_{kk'}]z_{1k}
\end{aligned}\tag{20}$$

In order to obtain a valid interpretation, z_{11} needs to be equal to 1. Symmetrically, the same calculation can be performed for the backward variable. The recurrence relations for the forward and backward variables are thus

$$\begin{cases} z_{11}=1\\ z_{1k'}=\sum_{k\in\mathrm{Pred}(k')}\exp[-\theta c_{kk'}]z_{1k}\end{cases}\qquad\begin{cases} z_{nn}=1\\ z_{kn}=\sum_{k'\in\mathrm{Succ}(k)}\exp[-\theta c_{kk'}]z_{k'n}\end{cases}\tag{21}$$

Eqs. (21) also hold for general graphs containing cycles but, in this case, the problem is no more multi-level, as for an acyclic lattice. Therefore, for general graphs with cycles, Eqs. (21) define two systems of linear equations that have to be solved (as shown in [49] by using a statistical physics framework). Although the idea of calculating the partition function in terms of recurrence relations is not new (see e.g., [70]) and is already computed with dynamic programming (see e.g., [40]), the above recurrence relations are valid not only for lattices, but also for any kind of graph.

Interestingly, these recurrence formulae are quite similar to the well-known forward–backward procedure appearing in hidden Markov models (HMM, see [44,43]). In fact, it can be shown that by assuming $\theta=1$ and defining $c_{kk'}=-\log(p_{kk'})$ (where $p_{kk'}$ are the transition probabilities) and by extending the lattice with "emission" nodes (modeling emission probabilities), the SoP formalism reduces to the forward/backward recurrences of the Baum–Welch algorithm for training hidden Markov models.

On the other hand, as noted by a reviewer, the SoP model is also closely related to the stochastic edit distance (SED) [6]. Indeed, the path probability proposed is approximately the same as the conventional likelihood of an alignment path $P(\mathbf{y}|\mathbf{x})$ between a given string, $\mathbf{x}$, and its "distorted version", $\mathbf{y}$, under the assumption that the (insertion, deletion, substitution) error probabilities are uniform. More precisely, let p_e be the probability of each of the (insertion, deletion, non-matching substitution) errors and p_m the probability for a matching substitution. The likelihood of $\mathbf{y}$ along path $\wp$ is the product of the probabilities of all the edit

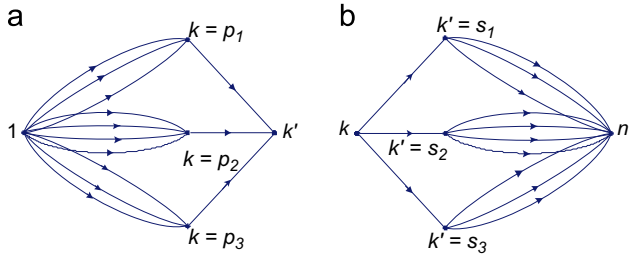

Fig. 3. (a) Forward recurrence: first compute the paths from node 1 to node k. Then, jump from node k to node k'. The nodes k are the predecessors of node k', $k\in\mathrm{Pred}(k')$. (b) Backward recurrence: first jump from node k to node k'. Then, compute the paths from node k' to node n. The nodes k' are the successors of node k, $k'\in\mathrm{Succ}(k)$.

operations along $\wp$, i.e., $P(\mathbf{y},\wp|\mathbf{x})=(p_e)^{n_e}(p_m)^{n_m}$ where n_e is the number of errors and n_m is the number of matches along $\wp$. If we define the related costs $c_e=-\log p_e$ and $c_m=-\log p_m$, $P(\mathbf{y},\wp|\mathbf{x})$ can be rewritten as $P(\mathbf{y},\wp|\mathbf{x})=\exp[-\sum_{k,k'\in\wp}c_{kk'}]=\exp[-C(\wp)]$, which turns out to be the same form as the SoP path probability with $\theta=1$. In that case, $\mathrm{P}(\mathbf{y}|\mathbf{x})=\sum_{\wp\in\mathcal{P}_{1n}}\mathrm{P}(\mathbf{y},\wp|\mathbf{x})=\sum_{\wp\in\mathcal{P}_{1n}}\exp[-C(\wp)]$, which corresponds to the partition function, $\mathcal{Z}$. Therefore, in this context, the partition function can be considered as a likelihood function, and $-\log\mathcal{Z}$ a dissimilarity measure between string $\mathbf{x}$ and string $\mathbf{y}$. This SED dissimilarity measure will be investigated in the experimental section.

2.5. The SoP longest common subsequence and extension to handle gaps

Let us now turn to the randomized version of the longest common subsequence (LCS). The dynamic programming lattice used by the LCS algorithm is in fact identical to that of the Levenshtein edit distance, and exactly the same SoP-based development can be applied, with only one (but important) difference: since LCS is a similarity measure, it involves immediate rewards or similarities $s_{kk'}$ instead of costs. Therefore, the probability distribution on the set of alignments is redefined as $P(\wp)=\exp[\theta S(\wp)]/\mathcal{Z}$ where $S(\wp)$ is the total similarity associated to path $\wp$ (the sum of the individual similarities along the path) and $\mathcal{Z}=\sum_{\wp\in\mathcal{P}_{1n}}\exp[\theta S(\wp)]$. Thus, good alignments (having a large similarity) are favored over bad ones (having a low similarity). The **SoP common subsequence similarity**, denoted as s_{SoP}, is defined as the average of $S(\wp)$ over the previous probability distribution. By proceeding exactly as in the previous section, we obtain

$$s_{\mathrm{SoP}}=\frac{1}{z_{1n}}\sum_{k=1}^{n}\sum_{k'=1}^{n}z_{1k}s_{kk'}\exp[\theta s_{kk'}]z_{k'n} \tag{22}$$

Notice that in the SoP CS case, as opposed to the SoP ED, cycles are not allowed, since the similarity could become arbitrarily high by looping.

It must be noticed that our model already handles gaps by allowing insertions and deletions, yet one may like to handle affine gaps [17]. Let us imagine that we have a lattice as the one in Fig. 4, where each node is represented twice to model the possibilities of continuing a gap, or starting a gap. The new graph has an increased number of edges in order to allow different costs depending whether we have already started the gap, or we start a new one (as we prefer fewer long gaps over many short ones). The number of possible paths remains the same as the possible editing operations

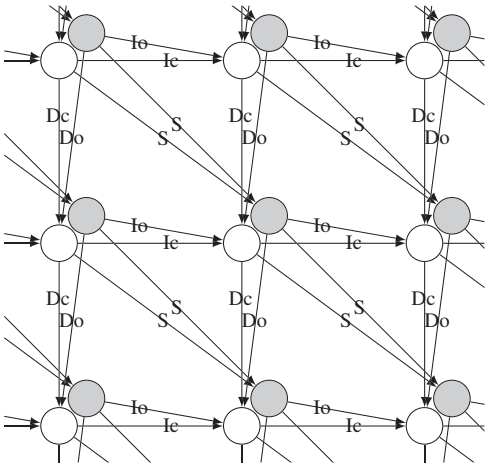

Fig. 4. Graph that allows affine gaps: the white nodes represent the continuation of a gap while grey nodes represent the start of a gap. DO/IO edges start a new gap, while DC/IC edges continue a gap.

are the same from any node, although with different costs. Furthermore, the expected cost remains the same when a linear gap model cost is assumed. For different affine gap models, from the linear model, the local costs would change letting the probability distribution vary and, therefore, the expected cost as well. It would be thus possible to integrate affine gaps in our solution and still use the same recurrence relations for computing the expected cost.

2.6. The SoP edit distance algorithm

The SoP edit distance can thus be computed as follows:

1. Compute the forward and backward variables using Eqs. (21).
2. Compute the SoP edit distance thanks to Eq. (19).

The details as well as the Matlab code of the SoP edit distance algorithm are available from www.isys.ucl.ac.be/staff/silvia. This algorithm depends on the parameter θ and assumes constant insertion and deletion costs, c_{ins} and c_{del}. The states of the dynamic programming lattice are indexed by $k:=(i,j)$. The insertion of a symbol corresponds to a transition $(i,j)\rightarrow(i,j+1)$ while the deletion of a symbol corresponds to $(i,j)\rightarrow(i+1,j)$. A substitution is $(i,j)\rightarrow(i+1,j+1)$ with cost $c_{\mathrm{sub}}(s_1(i),s_2(j))$. The first line ($j=1$) and column ($i=1$) correspond to a dummy, empty, symbol in the dynamic programming table. The forward and backward variables are denoted by z_{f} (for the z_{1k}) and z_{b} (for the z_{kn}), respectively.

3. Experiments

This experimental section aims at: (i) evaluating the relative performance of the two proposed methods: the SoP ED ($\mathbf{K}_{\mathrm{ED}}^{\mathrm{SoP}}$)[2] and the SoP CS ($\mathbf{K}_{\mathrm{CS}}^{\mathrm{SoP}}$), (ii) comparing their performance with their respective standard, non-randomized, versions: the LED ($\mathbf{K}_{\mathrm{LED}}$) and the LCS ($\mathbf{K}_{\mathrm{LCS}}$), and (iii) comparing their performance with state-of-the-art kernels used for sequence similarity computation [52] such as the ASK ($\mathbf{K}_{\mathrm{AS}}$) [65], the GASK ($\mathbf{K}_{\mathrm{GAS}}$) [35], the FLK ($\mathbf{K}_{\mathrm{FL}}$) [68], the PSPECK ($\mathbf{K}_{\mathrm{PSPEC}}$) [32], as well as the SED ($\mathbf{K}_{\mathrm{SED}}$) [6,41] and a method for biological sequence comparison, KD ($\mathbf{K}_{\mathrm{KD}}$) [67]. Notice that only four of these methods do not depend on a parameter (see Table 1 below).

Notice that the normalization and centering of the kernel matrices are also considered as meta-parameters (see Section 3.2 for more details). The local costs (or similarities) are set as follows: for edit distances $c_{ins}=c_{del}=c_{subs}=1$ (for non-matching substitutions) and $c_{subs}=0$ for matching substitutions; for common subsequences $s_{ins}=s_{del}=s_{subs}=0$ (for non-matching substitutions) and $s_{subs}=1$ for matching substitutions. Classification tasks on five real-world data sets have been performed as presented in the following sections. Notice also that in order to avoid overflow or underflow problems, we applied the standard formula for the logarithm of a sum (see, e.g., [24]) in computing z_{1k} and z_{kn}.

3.1. Data sets

This section introduces the different data sets used for the experiments and provide some examples (see Table 2) in order to give a more detailed insight of the performed experiments. Note that all the described data sets are made available in www.isys.ucl.ac.be/staff/silvia.

[2] Note that the two proposed methods are not valid kernels but similarity measures. However, for simplicity, the letter **K** will be used indifferently throughout the document.

Table 1
List of compared methods and parameter values tested in the internal cross-validation.

Algorithm and its abbreviations			Parameter	Tested values
$\mathbf{K}_{\mathbf{ED}}^{\mathbf{SoP}}$	SoP ED	Sum-over-paths edit distance	Degree of randomness θ	$\theta=\{0.1,0.5,1,1.5,2,2.5,3,3.5,4\}$
$\mathbf{K}_{\mathbf{CS}}^{\mathbf{SoP}}$	SoP CS	Sum-over-paths common subsequence	Degree of randomness θ	$\theta=\{0.1,0.5,1,1.5,2,2.5,3,3.5,4\}$
$\mathbf{K}_{\mathbf{SED}}$	SED	Stochastic edit distance	Error probability p_e	$p_e=\{0.05, 0.1, 0.15, 0.2, 0.25\}$
$\mathbf{K}_{\mathbf{LED}}$	LED	Levenshtein edit distance		
$\mathbf{K}_{\mathbf{LCS}}$	LCS	Longest common subsequence		
$\mathbf{K}_{\mathbf{KD}}$	KD	*K*-best distinct alignments		
$\mathbf{K}_{\mathbf{AS}}$	ASK	All subsequences kernel		
$\mathbf{K}_{\mathbf{GAS}}$	GASK	Gap-weighted all subsequences kernel	Weight of gaps λ	$\lambda=\{0.2,0.4,0.5,0.6,0.8\}$
$\mathbf{K}_{\mathbf{FL}}$	FLK	Fixed length kernel	Length of substring l	$l=\{5, 10, 15, 20\}$
$\mathbf{K}_{\mathbf{PSPEC}}$	PSPECK	*p*-Spectrum kernel	Length of subsequence p	$p=\{5, 10, 15, 20\}$

Table 2
Examples of samples of the tested data sets.

Data set	Data example	Example sequence representation
Digits	1	22444545545455544546667100110101101101011
Letters	b	82111111111111111111122699999999996333333332221111111144777778999
Figures	O	11111111112233333333333669999999999999999998877777777777111111111111
Arrows	←	1414444111144777777777111111111111111448
Proteins	Acetyltransferase	PAFYKKHGYKVIGVSEITPKGHNRYYLKKG

Digits data set: This data set was introduced in [41] and is originally based on the NIST Special Database 3 of the National Institute of Standards and Technology. A subset of 100 sequences of each digit (from 0 to 9), thus 1000 sequences in total from 10 different writers, was extracted for our experiments. Each sequence was obtained by mapping its shape from the bitmap digit to a sequence of numbers that expresses the direction of the perimeter. An example of these data can be seen in Table 2.

Letters, figures, and arrows data sets: These three data sets were collected [7] from 30 different people who wrote 10 times each letter from the English alphabet as well as a series of geometric shapes, such as arrows or other figures, on a digital tablet. Each of these three data sets is composed of sequences of numbers that represent the geographical directions of their perimeter as described in [7]. An example of these data can be seen in Table 2. For the *letters data set*, we selected the sequences corresponding to the subset of the first 13 letters of the alphabet {*a*,*b*,*c*,*d*,*e*,*f*,*g*, *h*,*i*,*j*,*k*,*l*,*m*}. This is a balanced data set with class priors of 7.7% each. The *arrows data set* corresponds to the set $\{\leftarrow,\uparrow,\rightarrow,\downarrow\}$ with a total number of sequences of 1010 where the class distribution is {256,252,245,257}. Sequences are extracted in the same way as for the letters data set. The last data set tested correspond to the *geometric figures* set. For this data set, 500 sequences comprising the following classes were collected: {*circle, triangle, rectangle, square, pentagon*}. This is a balanced set with class priors of 20% each. An example of these data can be seen in Table 2.

Proteins data set: This data set was collected from the UniProtKB and consists on five groups of similar proteins extracted via a BLAST query with default parameters on a *seed sequence*. The list of seed sequences as well as the proteins that integrate each of the groups are available on www.isys.ucl.ac.be/staff/silvia. Each class contains 50 similar sequences, and there are five classes in total, therefore providing a 250 balanced sequences data set.

3.2. Supervised classification methodology

Three types of supervised tasks have been performed over the whole data sets: (i) a 1-NN based on random prototypes, (ii) a 1-NN based on within-class centroids, and (iii) an SVM. In each case, a 10-fold nested cross-validation has been applied. Estimated classification rates as well as their 95% confidence interval are reported.

Normalization of kernel matrices: All calculated kernels or similarity measures are centered and/or then normalized. This normalization and/or centering are considered as another parameter to be tuned during internal cross-validation (four cases are tested: no centering nor normalization, centering only, normalization only, and centering and then normalization of the kernel). In this way, normalized (and/or centered) kernels are only used when their performance proves better than the one of the original kernel. For centering, we applied the transformation $\mathbf{K}_c=\mathbf{HKH}$ where $\mathbf{K}$ is the kernel or the similarity matrix, $\mathbf{H}=(\mathbf{I}-\mathbf{ee}^{\mathbf{T}}/n)$ is the centering matrix, $\mathbf{I}$ is the identity matrix, and $\mathbf{e}$ is a column vector full of ones. Notice that for $\mathbf{K}_{\mathrm{ED}}^{\mathrm{SoP}}$, $\mathbf{K}_{\mathrm{LED}}$ and $\mathbf{K}_{\mathrm{SED}}$, which are dissimilarity measures, the conversion to a similarity measure—as well as centering—is achieved through $\mathbf{K_c}=-\frac{1}{2}\mathbf{H}\Delta\mathbf{H}$ where Δ is the distance matrix containing the edit distances. This is a classical multidimensional scaling procedure [8,14]. After centering, all the obtained similarity matrices are further normalized with $\mathbf{K_n}=\mathbf{D}^{-1/2}\mathbf{K_c}\mathbf{D}^{-1/2}$ where $\mathbf{D}$ is a diagonal matrix containing the elements of the diagonal of $\mathbf{K_c}$, $\mathbf{D}=\mathbf{Diag}(\mathbf{K_c})$. We will refer to these transformed kernels (centered and/or normalized) as normalized kernels for the sake of simplicity.

Experimental methodology: The 1-NN based on random prototypes (1-NNP) assigns each observation to the class to which belongs his nearest prototype. A class prototype is randomly chosen among all observations from a given class, repeating this procedure with 10 different prototypes, therefore reporting an average of the estimated classification rates. In the case of the 1-NN based on within-class centroids (1-NNC), the observations are assigned to the class of its nearest centroid. In both cases, a double cross-validation was performed in order to tune both the parameter (see Table 1) as well as the kernel normalization choice, as explained above. This methodology also holds for the SVM,[3] where the additional parameter *C* had to be tuned. The tested values were

[3] The chosen implementation was the LIBLINEAR package [19] with default parameters. The package can be downloaded from http://www.csie.ntu.edu.tw/~cjlin/liblinear.

$C=\{0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000\}$, and a grid cross-validation was used in this case.

For this supervised classification task, simple nearest neighbor rules are used because the aim of the experiments, and more generally the work, is not to design a state-of-the-art sequence classifier; they rather aim at comparing different similarity measures between sequences. A good similarity measure should lead to compact, well-separated, classes in the embedding space. This is also the reason why we tested both the within-class centroids, and the random prototypes as class representative. A crude classifier, like the 1-NN we are using, will probably be more sensitive to the compactness/separability of the different classes in the embedding space while a more sophisticated classifier, like a SVM, could achieve good performance, even if the different classes are not well-separated and compact.

Table 3
p-Values for paired sign test for each classification method and algorithm.

Classification method	SoP ED vs. LED	SoP CS vs. LCS	SoP CS vs. SoP ED
1-NN based on within-class centroids	0.0000	0.0147	0.0000
1-NN based on prototypes	0.0000	0.0000	0.0000
SVM	0.0551	0.0266	0.1537

3.3. Results and test of hypothesis

Results obtained by nested cross-validation for all methods on all data sets are reported in Table 4. In order to verify that the

Table 4
Estimated classification rates with a 95% confidence interval obtained with the three classification methods. Best performing results are highlighted in grey, and second-best methods are marked in bold. The results marked with ∗ have been omitted due to excessive computation time.

Classification results (1-NN using random class prototypes)

	Arrows Data Set	Digits Data Set	Figures Data Set	Letters Data Set	Proteins Data Set
$\mathbf{K}_{\mathbf{ED}}^{\mathbf{SoP}}$	73.59 ± 0.34%	66.37 ± 0.37%	35.10 ± 0.39%	**40.18 ± 0.22%**	**39.28 ± 0.53%**
$\mathbf{K}_{\mathbf{LED}}$	68.68 ± 0.88%	61.46 ± 0.36%	35.08 ± 0.22%	34.20 ± 0.21%	39.72 ± 0.83%
$\mathbf{K}_{\mathbf{SED}}$	68.76 ± 0.52%	63.35 ± 0.43%	34.38 ± 0.31%	36.05 ± 0.12%	37.88 ± 0.68%
$\mathbf{K}_{\mathbf{CS}}^{\mathbf{SoP}}$	81.26 ± 0.44%	72.59 ± 0.42%	37.28 ± 0.32%	45.26 ± 0.24%	37.28 ± 0.45%
$\mathbf{K}_{\mathbf{LCS}}$	**79.61 ± 0.71%**	**67.10 ± 0.34%**	**37.00 ± 0.29%**	39.61 ± 0.22%	34.96 ± 0.55%
$\mathbf{K}_{\mathbf{AS}}$	76.63 ± 0.83%	31.00 ± 0.36%	25.88 ± 0.47%	18.39 ± 0.63%	27.72 ± 0.47%
$\mathbf{K}_{\mathbf{FL}}$	72.79 ± 0.86%	32.34 ± 0.52%	28.58 ± 0.37%	22.79 ± 0.27%	26.60 ± 0.25%
$\mathbf{K}_{\mathbf{PSPEC}}$	68.32 ± 0.62%	33.57 ± 0.31%	34.86 ± 0.51%	16.54 ± 0.14%	36.56 ± 0.54%
$\mathbf{K}_{\mathbf{KD}}$	43.71 ± 0.42%	17.69 ± 0.35%	35.98 ± 0.64%	20.95 ± 0.33%	32.56 ± 0.59%
$\mathbf{K}_{\mathbf{GAS}}$	78.81 ± 0.53%	58.31 ± 0.30%	(*)	32.08 ± 0.33%	39.48 ± 0.54%

Classification results (1-NN using within-class centroids)

	Arrows Data Set	Digits Data Set	Figures Data Set	Letters Data Set	Proteins Data Set
$\mathbf{K}_{\mathbf{ED}}^{\mathbf{SoP}}$	96.34 ± 1.53%	91.60 ± 1.61%	56.80 ± 2.63%	70.96 ± 2.56%	71.20 ± 4.93%
$\mathbf{K}_{\mathbf{LED}}$	96.34 ± 1.64%	89.40 ± 1.90%	53.60 ± 1.92%	67.31 ± 2.69%	68.00 ± 4.53%
$\mathbf{K}_{\mathbf{SED}}$	96.04 ± 1.69%	90.60 ± 2.23%	53.00 ± 2.13%	68.94 ± 2.70%	66.00 ± 4.71%
$\mathbf{K}_{\mathbf{CS}}^{\mathbf{SoP}}$	**96.73 ± 1.78%**	95.00 ± 1.20%	63.60 ± 1.52%	77.69 ± 2.21%	**71.20 ± 5.07%**
$\mathbf{K}_{\mathbf{LCS}}$	97.13 ± 1.70%	**94.20 ± 1.20%**	**59.80 ± 2.22%**	**77.12 ± 1.45%**	70.00 ± 4.41%
$\mathbf{K}_{\mathbf{AS}}$	25.64 ± 0.79%	10.00 ± 0.00%	20.00 ± 0.00%	7.69 ± 0.00%	20.80 ± 1.05%
$\mathbf{K}_{\mathbf{FL}}$	74.26 ± 1.83%	46.40 ± 2.25%	29.60 ± 1.52%	20.58 ± 1.87%	28.80 ± 3.05%
$\mathbf{K}_{\mathbf{PSPEC}}$	80.59 ± 4.10%	54.70 ± 1.92%	43.00 ± 3.21%	24.52 ± 1.60%	58.40 ± 6.10%
$\mathbf{K}_{\mathbf{KD}}$	50.20 ± 2.15%	23.90 ± 2.05%	53.20 ± 2.81%	38.08 ± 2.52%	66.80 ± 5.49%
$\mathbf{K}_{\mathbf{GAS}}$	26.63 ± 0.94%	11.70 ± 0.59%	(*)	7.69 ± 0.00%	23.20 ± 2.28%

Classification results (SVM)

	Arrows Data Set	Digits Data Set	Figures Data Set	Letters Data Set	Proteins Data Set
$\mathbf{K}_{\mathbf{ED}}^{\mathbf{SoP}}$	98.51 ± 1.05%	97.90 ± 1.07%	**84.80 ± 1.46%**	**94.04 ± 0.78%**	73.20 ± 4.68%
$\mathbf{K}_{\mathbf{LED}}$	**98.61 ± 0.88%**	97.90 ± 1.03%	83.80 ± 1.23%	92.12 ± 1.75%	68.00 ± 6.61%
$\mathbf{K}_{\mathbf{SED}}$	98.32 ± 1.16%	97.40 ± 0.98%	87.80 ± 1.98%	94.42 ± 0.92%	**73.60 ± 4.85%**
$\mathbf{K}_{\mathbf{CS}}^{\mathbf{SoP}}$	98.71 ± 1.00%	**98.00 ± 1.05%**	83.00 ± 2.76%	93.56 ± 0.93%	74.80 ± 5.23%
$\mathbf{K}_{\mathbf{LCS}}$	98.22 ± 1.03%	98.30 ± 1.01%	83.80 ± 2.44%	91.83 ± 1.23%	70.40 ± 4.56%
$\mathbf{K}_{\mathbf{AS}}$	92.28 ± 1.07%	66.20 ± 4.83%	20.00 ± 0.00%	26.92 ± 4.68%	23.60 ± 4.44%
$\mathbf{K}_{\mathbf{FL}}$	96.24 ± 1.35%	80.30 ± 5.45%	76.20 ± 2.06%	72.31 ± 3.09%	36.00 ± 6.51%
$\mathbf{K}_{\mathbf{PSPEC}}$	97.52 ± 1.45%	77.80 ± 3.16%	59.80 ± 6.44%	59.33 ± 1.97%	68.40 ± 5.90%
$\mathbf{K}_{\mathbf{KD}}$	95.45 ± 1.20%	82.10 ± 1.86%	81.40 ± 3.51%	84.62 ± 2.37%	67.20 ± 5.82%
$\mathbf{K}_{\mathbf{GAS}}$	95.84 ± 1.19%	83.60 ± 8.54%	(*)	54.04 ± 3.83%	41.20 ± 4.68%

proposed methods, SoP ED and SoP CS, perform better than their non-randomized versions, LED and LCS, a non-parametric one-sided paired sign test has been performed. The classification rates from the classification experiments for all data folds have been used as sample for this test (10-folds $\times$ 5 data sets = 50 samples per classification method and algorithm). Results showed that, with a 95% confidence level, the proposed methods perform better in all cases, except in the case of the SVM for the SoP ED, where the confidence is 94%. Moreover, the SoP CS appears to perform slightly better than the SoP ED (with 100% confidence level for the 1-NN and 84% for the SVM). The obtained p-values are shown in Table 3. While significant, the magnitude of the improvement is, however, not always spectacular on these investigated data sets (see Table 4).

3.4. Discussion of the results

The conclusions following this experimental study are rather clear; let us indeed return to our research questions. First, we can conclude that, in general, the SoP CS similarity measure and the SoP ED similarity measure based on the edit distance usually perform significantly better than the original standard measures (standard LED and LCS), as shown by the test of hypothesis. It can be observed that when LCS performs better than LED, it is the randomized version, SoP CS, which will perform best (respectively for the LED and SoP ED). Second, it has been observed that the SoP methods show an improvement over the remaining methods, be it on the OCR classification tasks or on the protein task. Third, the string kernels tested in this paper (the all subsequences kernel, the gap-weighted all subsequences kernel, the fixed length kernel, and the p-spectrum kernel) performed poorly in comparison with, for instance, a standard longest common subsequence measure. Here is a tentative explanation of this fact. As discussed in the related work from Section 1.2, we expect string kernels to perform well when the alphabet is very large (for instance in the context of text mining—in this case—there are very few long common subsequences), and worse for reduced alphabets. It must also be noted the good performance of the stochastic edit distance (SED) in the case of the SVM for the letters and figures data sets. These two data sets contain the longest sequences among all tested data. This may indicate that the SED is a more performing measure for long sequence comparison combined with an SVM classifier. Yet another important property of the SED is that it is based on a log-likelihood instead of an expected cost (or similarity) as the SoP ED and CS (see the discussion end of subsection 2.4). Finally, there is no clear winning string kernel for all the tested data sets.

In conclusion, these experiments show that the best method overall are the sum-over-paths distances. They consistently provide good results, they are almost always among the best method and, when is not the case, their performance is always very close to the winning method. Moreover, their computation complexity is similar to that of the standard edit distance and longest common subsequence algorithms, although at the price of a parameter, θ. Finally, the SoP common subsequence seems to provide slightly better results than the SoP edit distance. It must also be noticed that the spatial complexity could be improved by applying the *Four Russians* method [5], although there is a trade-off between the spatial and time complexity of the algorithm.

4. Conclusion

This work proposed a procedure for randomizing a dynamic programming algorithm defined on a lattice. It first defines a Boltzmann probability distribution on the set of paths through the lattice in such a way that good paths have a high probability of occurrence while bad paths have a low probability of being followed. Then, instead of computing the dynamic programming score on the optimal paths only, it averages the scores along all the possible paths, each individual score along a path being weighted by the probability of following it. This allows to account for the contributions of good, although sub-optimal, paths as well. Forward/backward recurrence relations allowing efficiently computation of the score are developed along the same line as the forward/backward algorithm in hidden Markov models.

Experimental results obtained on sequence classification tasks indicate that, in some cases, taking the suboptimal alignments into account in the expected cost, improves significantly the results. In the remaining situations, it is the standard versions that perform best (the LED and the LCS). It must be noticed that in that case, its randomized version is the second winning method, and if we had extended the range for θ values, the performance would have been the same (as the standard measure is a specific case of the randomized one when $\theta \to \infty$).

Future work will be devoted to the development and the assessment of randomized fractional programming techniques allowing to optimize the ratio of two quantities along a path, for instance the normalized score—the total score along a path divided by its length—which leads to the normalized edit distance [37,62]. Another interesting extension would be to derive a valid positive semi-definite edit distance kernel based on the same ideas.

Acknowledgments

This work is partially supported by the "Fonds pour la formation à la Recherche dans l'Industrie et dans l'Agriculture" (F.R.I.A.) under Grant reference F3/5/5-MCF/ROI/BC-21716. Part of this work has also been funded by projects with the "Région Wallonne" and the Belgian "Politique Scientifique Fédérale". We thank these institutions for giving us the opportunity to conduct both fundamental and applied research. We would like to thank as well Professor Marc Sebban from the University of Jean Monnet, France, for providing us with the digits data set used in the experiments. We also thank Amin Mantrach for his help with the experiments and Eric Vandenbussche for his help in proving that the recurrence relations still hold in the case of graphs including cycles. Both are from the Université libre de Bruxelles, Belgium.

References

[1] Y. Achbany, F. Fouss, L. Yen, A. Pirotte, M. Saerens, Optimal tuning of continual exploration in reinforcement learning, in: Proceedings of the 16th International Conference on Artificial Neural Networks (ICANN 06), Lecture Notes in Computer Science, vol. 4131, 2006, pp. 734–749.

[2] Y. Achbany, F. Fouss, L. Yen, A. Pirotte, M. Saerens, Tuning continual exploration in reinforcement learning: an optimality property of the Boltzmann strategy, Neurocomputing 71 (2008) 2507–2520.

[3] T. Akamatsu, Cyclic flows, Markov process and stochastic traffic assignment, Transportation Research B 30 (5) (1996) 369–386.

[4] Juan-Carlos Amengual and Enrique Vidal. On the estimation of error-correcting parameters, in: Proceedings of the International Conference on Pattern Recognition (ICPR), 2000, pp. 2883–2886.

[5] V.L. Arlazarov, E.A. Dinic, M.A. Kronod, I.A. Faradzev, On economical construction of the transitive closure of an oriented graph, Doklady Akademii Nauk SSSR 194 (1970) 487–488.

[6] L.R. Bahl, F. Jelinek, Decoding for channels with insertions, deletions, and substitutions with applications to speech recognition, IEEE Transactions on Information Theory 21 (4) (1975) 404–411.

[7] F. Beuvens, T. Dullier, Développement et expérimentation d'une plate-forme de reconnaissance de gestes par stylet, Master's Thesis, Université Catholique de Louvain, 2009.

[8] I. Borg, P. Groenen, Modern Multidimensional Scaling: Theory and Applications, Springer, New York, 1997.

[9] S. Boyd, P. Diaconis, L. Xiao, Fastest mixing Markov chain on a graph, SIAM Review (2004) 667–689.

[10] P. Bucher, K. Hofmann, A sequence similarity search algorithm based on a probabilistic interpretation of an alignment scoring system, in: Proceedings of the 4th International Conference on Intelligent Systems for Molecular Biology (ISMB 1996), AAAI, 1996.

[11] N. Cancedda, E. Gaussier, C. Goutte, J.M. Renders, Word sequence kernels, Journal of Machine Learning Research 3 (2003) 1059–1082.
[12] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., MIT Press, McGraw-Hill Book Company, Cambridge, Massachusetts, 2000.
[13] T.M. Cover, J.A. Thomas, Elements of Information Theory, John Wiley & Sons, New Jersey, 1991.
[14] T. Cox, M. Cox, Multidimensional Scaling, second ed., Chapman and Hall, Boca Raton, Floride, 2001.
[15] J.-C. Delvenne, A.-S. Libert, Centrality measures and thermodynamic formalism for complex networks, Manuscript, submitted for publication.
[16] C.B. Do, S.S. Gross, S. Batzoglou, CONTRAlign: discriminative training for protein sequence alignment, in: Proceedings of the 10th Annual International Conference on Computational Molecular Biology (RECOMB 2006), 2006.
[17] R. Durbin, S.R. Eddy, A. Krogh, G. Mitchison, Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, Cambridge University Press, Cambridge, 1998.
[18] L. Ekroot, T. Cover, The entropy of Markov trajectories, IEEE Transactions on Information Theory 39 (4) (1993) 1418–1421.
[19] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, C.-J. Lin, LIBLINEAR: a library for large linear classification, Journal of Machine Learning Research 9 (2008) 1871–1874.
[20] G.D. Fornay, The Viterbi algorithm, Proceedings of the IEEE 61 (3) (1973) 268–278.
[21] V. Girardin, Entropy minimization for Markov and semi-Markov processes, Methodology and Computing in Applied Probability 6 (2004) 109–127.
[22] V. Girardin, N. Limnios, Entropy rate and maximum entropy methods for countable semi-Markov chains, Communications in Statistics 33 (3) (2004) 609–622.
[23] D. Gusfield, Algorithms on Strings, Trees, and Sequences, Cambridge University Press, Cambridge, 1997.
[24] X. Huang, Y. Ariki, M. Jack, Hidden Markov Models for Speech Recognition, Columbia University Press, New York, NY, USA, 1990.
[25] T. Hwa, M. Lässig, Similarity detection and localization, Physical Review Letters 76 (1996) 2591.
[26] T. Jaakkola, M. Diekhans, D. Haussler, A discriminative framework for detecting remote protein homologies, Journal of Computational Biology 7 (1–2) (2000) 95–114.
[27] D. Jurafsky, J.H. Martin, Speech and Language Processing, second ed., Prentice Hall, Upper Saddle River, New Jersey, 2008.
[28] A. Krogh, M. Brown, I.S. Mian, K. Sjölander, D. Haussler, Hidden Markov models in computational biology: applications to protein modeling, Journal of Molecular Biology 235 (5) (1994) 1501–1531.
[29] J.B. Kruskal, An overview of sequence comparison: time warps, string edits, and macromolecules, SIAM Review 25 (1983) 201–237.
[30] M. Kschischo, M. Lässig, Finite-temperature sequence alignment, Pacific Symposium on Biocomputing 5 (2000) 624–635.
[31] C.S. Leslie, E. Eskin, A. Cohen, J. Weston, W. Stafford Noble, Mismatch string kernels for discriminative protein classification, Bioinformatics 20 (4) (2004) 467–476.
[32] C.S. Leslie, E. Eskin, W.S. Noble, The spectrum kernel: a string kernel for SVM protein classification, in: Pacific Symposium on Biocomputing, 2002, pp. 566–575.
[33] L. Liao, W. Stafford Noble, Combining pairwise sequence similarity and support vector machines for remote protein homology detection, in: RECOMB '02: Proceedings of the 6th Annual International Conference on Computational Biology, ACM, 2002, pp. 225–232.
[34] M.L. Littman, Markov games as a framework for multi-agent reinforcement learning, in: Proceedings of the 11th International Conference on Machine Learning (ICML-94), 1994, pp. 157–163.
[35] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, C. Watkins, B. Scholkopf, Text classification using string kernels, Journal of Machine Learning Research 2 (2002) 563–569.
[36] A. Mantrach, L. Yen, J. Callut, K. Françoisse, M. Shimbo, M. Saerens, The sum-over-paths covariance kernel: a novel covariance measure between nodes of a directed graph, IEEE Transactions Pattern Analysis and Machine Intelligence 32 (6) (2010) 1112–1126.
[37] A. Marzal, E. Vidal, Computation of normalized edit distance and applications, IEEE Transactions on Pattern Analysis and Machine Intelligence 15 (9) (1993) 926–932.
[38] S. Miyazawa, A reliable sequence alignment method based on probabilities of residue correspondences, Protein Engineering (8) (1995) 999–1009.
[39] G. Navarro, A guided tour to approximate string matching, ACM Computing Surveys 33 (1) (2001) 31–88.
[40] L.A. Newberg, C.E. Lawrence, Exact calculation of distributions on integers, with application to sequence alignment, Journal of Computational Biology: A Journal of Computational Molecular Cell Biology 16 (1) (2009) 1–18.
[41] J. Oncina, M. Sebban, Learning stochastic edit distance: application in handwritten character recognition, Pattern Recognition 39 (9) (2006) 1575–1587.
[42] M.J. Osborne, An Introduction to Game Theory, Oxford University Press, Oxford, 2004.
[43] L. Rabiner, B. Hwang Juang, Fundamentals of Speech Recognition, Prentice Hall PTR, Englewood Cliffs, New Jersey, April 1993.
[44] L. Rabiner, Readings in Speech Recognition. chapter A tutorial on hidden Markov models and selected applications in speech recognition, Morgan Kaufmann Publishers Inc., 1990, pp. 267–296.
[45] E. Ricci, T. De Bie, N. Cristianini, Learning to align: a statistical approach, in: Proceedings of the 7th International Symposium on Intelligent Data Analysis (IDA 2007), Ljubljana, 2007.
[46] E. Ristad, P. Yianilos, Learning string-edit distance, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (5) (1998) 522–532.
[47] E. Sven Ristad, P.N. Yianilos, Learning string-edit distance, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (5) (1998) 522–532.
[48] J. Rousu, J. Shawe-Taylor, Efficient computation of gapped substring kernels on large alphabets, Journal of Machine Learning Research 6 (2005) 1323–1344.
[49] M. Saerens, Y. Achbany, F. Fouss, L. Yen, Randomized shortest-path problems: two related models, Neural Computation 21 (8) (2009) 2363–2404.
[50] H. Saigo, J.-P. Vert, N. Ueda, T. Akutsu, Protein homology detection using string alignment kernels, Bioinformatics 20 (11) (2004) 1682–1689.
[51] D. Sankoff, J.B. Kruskal, Time Warps, String Edits, and Macromolecules, Addison-Wesley, Stanford, California, 1983.
[52] J. Shawe-Taylor, N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge University Press, Cambridge, 2004.
[53] M. Steel, J. Hein, Applying the Thorne–Kishino–Felsenstein model to sequence evolution on a star-shaped tree, Applied Mathematics Letters (14) (2001) 679–684.
[54] G. Stephen, String Searching Algorithms, World Scientific, Singapore, 1994.
[55] J. Sun, S. Boyd, L. Xiao, P. Diaconis, The fastest mixing Markov process on a graph and a connection to a maximum variance unfolding problem, SIAM Review (2006) 681–699.
[56] A. Tahbaz, A. Jadbabaie, A one-parameter family of distributed consensus algorithms with boundary: from shortest paths to mean hitting times, in: Proceedings of IEEE Conference on Decision and Control, 2006, pp. 4664–4669.
[57] S. Theodoridis, K. Koutroumbas, Pattern Recognition, third ed., Academic Press, 2006.
[58] J.L. Thorne, H. Kishino, J. Felsenstein, An evolutionary model for maximum likelihood alignment of dna sequences, Journal of Molecular Evolution 33 (1991) 114–124.
[59] J.L. Thorne, H. Kishino, J. Felsenstein, Inching toward reality: an improved likelihood model of sequence evolution, Journal of Molecular Evolution (34) (1992) 3–16.
[60] E. Todorov, Linearly-solvable Markov decision problems, in: Advances in Neural Information Processing Systems 19 (NIPS 2006), MIT Press, 2006, pp. 1369–1375.
[61] J. Tomlin, A new paradigm for ranking pages on the world wide web, in: Proceedings of the International World Wide Web Conference (WWW2003), 2003, pp. 350–355.
[62] E. Vidal, A. Marzal, P. Aibar, Fast computation of normalized edit distances, IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (9) (1995) 899–902.
[63] S.V.N. Vishwanathan, K.M. Borgwardt, R.I. Kondor, N.N. Schraudolph, Graph kernels, Journal of Machine Learning Research 11 (2010) 1201–1242.
[64] S.V.N. Vishwanathan, K.M. Borgwardt, N.N. Schraudolph, Fast computation of graph kernels, in: Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference, MIT Press, Cambridge MA, USA, 2007.
[65] S.V.N. Vishwanathan, A.J. Smola, Fast kernels for string and tree matching, in: Advances in Neural Information Processing Systems 15, MIT Press, 2003, pp. 569–576.
[66] R.A. Wagner, M.J. Fischer, The string-to-string correction problem, Journal of the ACM 21 (1) (1974) 168–173.
[67] M.S. Waterman, M. Eggert, A new algorithm for best subsequence alignments with application to trna-rrna comparisons, Journal of Molecular Biology 197 (4) (1987) 723–728.
[68] C. Watkins, Kernels from matching operations, Technical Report, Department of Computer Science, Royal Holloway, University of London, 1999.
[69] L. Yen, A. Mantrach, M. Shimbo, M. Saerens, A family of dissimilarity measures between nodes generalizing both the shortest-path and the commute-time distances, in: Proceedings of the 14th SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008, pp. 785–793.
[70] M.Q. Zhang, T.G. Marr, Alignment of molecular sequences seen as random path analysis, Journal of Theoretical Biology 174 (2) (1995) 119–129.

Silvia García-Díez received both the B.Sc. degree and the M.Sc. degree in Computer Engineering from the Universidad de Valladolid (UVA). She then became research assistant in the Université Catholique de Louvain (UCL) and is currently doing a Ph.D. degree in Applied Sciences thanks to a F.R.I.A. grant. Some of her research interests include data mining, machine learning and pattern recognition applied to music processing.

François Fouss received the Ph.D. degree in Management Science in 2007, from the Université catholique de Louvain (UCL), Belgium. In 2007, he joined the Facultés Universitaires Catholiques de Mons (FUCaM) as a professor in Computing Science. His main research areas include collaborative recommendations, graph mining, and classification.

Masashi Shimbo received the Ph.D. degree in Engineering from Kyoto University in 2000. He is currently an assistant professor in the Graduate School of Information Science, Nara Institute of Science and Technology. His research interests include graph-theoretic approaches to data mining and application of machine learning techniques to natural language processing.

Marco Saerens received the B.Sc. degree in Physics Engineering and the M.Sc. degree in Theoretical Physics, all from the Université Libre de Bruxelles (ULB). After graduation, he joined the IRIDIA Laboratory (the artificial intelligence laboratory, Université Libre de Bruxelles (ULB), Belgium) as a research assistant and completed the Ph.D. degree in Applied Sciences. While remaining a part-time researcher at IRIDIA, he then worked as a senior researcher in the R&D department of various companies, mainly in the fields of speech recognition, data mining, and artificial intelligence. In 2002, he joined the Université catholique de Louvain (UCL) as a professor in Computer Sciences. His main research interests include artificial intelligence, machine learning, pattern recognition, data mining, and speech/language processing.