Universitat d'Alacant
Universidad de Alicante

Departamento de Lenguajes y Sistemas Informáticos
Escuela Politécnica Superior

Similarity Learning and Stochastic Language Models for Tree-Represented Music

Jose Francisco Bernabeu Briones

Tesis presentada para aspirar al grado de

DOCTOR POR LA UNIVERSIDAD DE ALICANTE

DOCTORADO EN INFORMÁTICA

Dirigida por

Dr. José Manuel Iñesta Quereda
Dr. Jorge Calera Rubio

Para Carla, Chloe, Rocío y Neo,
por sus risas y ocurrencias,
además de su infinito cariño.

Agradecimientos

Antes de empezar decir que han sido muchas las personas a las que quisiera dar las gracias, si alguien se me olvida estoy seguro de que lo entenderá.

Aun recuerdo a mi abuelo calentando un clavo hasta ponerlo incandescente con el objetivo de hacer unos agujeros en un bote de plástico vacío, donde antes había helado. Seguidamente pasaba un cordel por esos agujeros y finalmente me lo colgaba a mi, obsequiándome con un par de palos, cortados de forma similar, procedentes de una escoba vieja. Posiblemente aquel fue el inicio de la pasión que siento por la batería en particular y la música en general. Sin duda alguna, para él queda una mención especial en esta tesis, por todo lo que me enseñó y por todos aquellos momentos que pasamos juntos.

Como otro acontecimiento importante queda el día en que mis padres tuvieron que salir a una reunión del colegio y yo me tuve que quedar sólo en casa. No os podéis imaginar la cara que pusieron cuando regresaron y se encontraron todas las cajas de zapatos puestas por el sofá mientras yo apaleaba aquellas cajas y el mismo sofá con un par de agujas de punto. Por esa y otras tantas trastadas que me han aguantado además de ayudarme y apoyarme siempre en todas mis locuras. A ellos junto a mi hermano pequeño, muchas gracias por todo.

En cuanto a mis dos directores. Sin duda alguna, la persona que consiguió fusionar mi pasión por la música y fascinación por el aprendizaje y la resolución de problemas ha sido José Manuel Iñesta. Quiero agradecerle desde aquí, tanto sus aportaciones a esta tesis como el darme la oportunidad de entrar a formar parte del Grupo de Reconocimiento de Formas e Inteligencia Artificial del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante, así como su infinita paciencia. Y a Jorge Calera, por resolver todas mis dudas y responder a todas mis preguntas acerca de la tesis, además de enseñarme miles de cosas en paralelo a la realización de esta tesis.

Agradecer, en especial, a los que hemos formado el grupo de jotas de estos últimos años Jose Oncina (gracias también por llevarme de paseo), Jorge Calvo, Jose Javier Valero (gracias por esos ánimos y por acompañarme en todos los papeleos) y José Manuel Iñesta viéndonos las caras de sueño y contando las batallas mañaneras. Que haría yo sin ellas! Por supuesto a Luisa Micó, a la que echábamos mucho de menos en esos desayunos, por ayudarme y aconsejarme siempre que ha sido necesario.

Gracias a David Rizo por poner el punto de inicio de esta tesis y contagiarme de su entusiasmo y su trabajo en este campo. Sin duda alguna, sin David esta tesis no creo que hubiese ni siquiera nacido.

Miles de gracias a este señor que tengo detrás de mi, Javier Sober . Muchas gracias por esas risas que nos pegamos y por no poner muy fuerte el aire acondicionado ;-)

Y en general agradecer a todo el Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante por haberme proporcionado un ambiente de trabajo envidiable. En especial a mis compañeros del Grupo de Reconocimiento de Formas e Inteligencia Artificial, Pierre León, Antonio Pertusa, Carlos Pérez-Sancho, Tomás Pérez, Javi Gallego, Juan Ramón Rico, José Luis Verdu, Alicia Garrido, Paco Moreno, Eva Gómez, porque de una manera o de otra me habéis ayudado a realizar esta tesis.

A Leo Becerra, Aurélien Bellet, Marc Sebban y Amaury Habrard por su gran hospitalidad cuando estuve de estancia en el Laboratoire Hubert Curien UMR 5516, F-42023, Saint-Etienne, France.

El conjunto de publicaciones que soportan este trabajo de investigación ha sido financiado a lo largo de los años por los proyectos:

- TIMuL: Tecnologías interactivas para el aprendizaje de música. Ministerio de Economía y Competitividad. TIN2013-48152-C2-1-R.
- DRIMS: Descripción y recuperación de información musical y sonora. Ministerio de Ciencia e Innovación. TIN2009-14247-C02-02.
- PASCAL2: Pascal European Network of Excellence. VII Framework Program.
- MIPRCV: Multimodal Interaction in Pattern Recognition and Computer Vision. Ministerio de Educación y Ciencia (Consolider Ingenio 2010). CSD2007-00018.
- PROSEMUS: Procesamiento semántico de música digital. Ministerio de Educación y Ciencia. TIN2006-14932-C02-02.

Agradecer también desde aquí a los amigachos Jijonencos de toda la vida así como a mis compañeros y amigos en Stoneheads, RadioZ y la familia Mind's Doors. Gracias por esa paciencia y esos buenos ratos que pasamos juntos. En especial a César Alcaráz, cuyos consejos y forma de ver la vida siempre ha sido de gran ayuda.

Por último, el mayor agradecimiento va para mi familia (iaios, abuelos, tios, tias, cuñados, cuñadas, etc, ...), por estar ahí y ayudarnos siempre que ha sido necesario. Especialmente para mi gran amor Rocío, y las dos princesas Carla y Chloe. Ellas tres juntas hacen que cada día sea una experiencia nueva llena de risas, amor y buenos momentos que guardo en lo más profundo de mi ser. Gracias por estar ahí!

Contents

List of Figures

List of Tables

1 Introduction

One of the main concerns in music information retrieval (MIR) tasks is how to assess melodic similarity in a similar way to how humans do. We are very good at recognizing previously known patterns, even if our perceived inputs are distorted, they are presented just partially, or in the presence of noisy data. This happens in music comparison in a number of situations, for example, when comparing different cover versions of a given melody or when searching in databases using a query that will be, by its own definition, partial and can be distorted or even wrong. Two issues are concerned to this problem: the similarity computation and the representation structure.

The term *music similarity* is ambiguous or at least it can be judged from different points of view (Selfridge-Field, 1998). It may refer to the resemblance between the melodic line of two musical fragments, the similarity of their rhythmic patterns, or even their harmonic coincidence. In order to disambiguate this concept for the span of this dissertation, we will consider as ground-truth that the most similar sequences are different interpretations of the same song or those produced by the variation compositional form. This statement implies that the similarity is measured upon a trade-off between the melodic and rhythmic dimensions.

1.1 Music Information Retrieval (MIR)

Music information retrieval (MIR) is a field of research devoted to the extraction of meaningful information from the content of music sources (Orio, 2006), (Typke et al., 2005).

Traditionally, this research has been divided into the audio and symbolic domains. Digital audio files contain a digitized audio signal coming from a sound recording, and can be found in compressed (MP3) or uncompressed (WAV) formats. On the other hand, symbolic music files contain digital scores, i.e. the music notation and instructions necessary for a computer

or synthesizer to play the song. These files can be sequenced (MIDI) or structurally represented (MusicXML).

With the widespread use of portable music players, most computer users today have to deal with large digital music database. Plenty of software tools (e.g. iTunes) are available to play, retrieve, and store huge quantities of audio files. These files contain richer information than symbolic ones, because they bring together all the different elements that take part in a musical work: pitch, harmony, rhythm, timbre, lyrics, etc. However, this richness results in a bigger complexity, because all this information is mixed in the audio signal and it is very difficult to isolate each one of these elements from the others. For working with audio files it is necessary to use digital signal processing techniques in order to extract some features representing the musical content. For example, one of the most common features used for this purpose are Mel frequency cepstral coefficients, which provide information on the timbre of the piece (Aucouturier and F., 2004). Other works use features regarding rhythm (Lidy and Rauber, 2005), texture (Tzanetakis and Cook, 2002), or pitch (Tzanetakis et al., 2003). However, these features only provide unaccurate descriptions of the musical content and are difficult to interpret.

Because of the sudden huge popularity of portable audio players, research about algorithms that sort, retrieve, and suggest music have become really important recently. In this framework, arises the need to transform the audio files in a format that provide very accurate information on the actual content of a musical work - the score - including pitch, harmony, rhythm, and lyrics, but lack some important features that can be only found in the audio files, such as timbre and performance and production issues. The format which get this is symbolic music files (data), and the process to transform audio data to symbolic data is the transcription.

However, state-of-the-art transcription algorithms are not reliable today. Hence, most of the music models today do not consider the musical structure at all. They mostly rely on local properties of audio signal, such as texture, or short term frequency analysis. For instance, in most current approaches for transcription or pitch tracking, the algorithms have to rely on strong assumptions about timbre or the number of simultaneous notes to decide how many notes are simultaneously played, and to identify these notes. The general applicability of these algorithms is thus limited.

However, very little research has been done to model symbolic music data compared to the important efforts deployed to model audio data. There are some works that use symbolic data to improve the results using only audio data. An example of this is the approach in the context of genre classification is proposed by (Lidy T., 2007) where audio features and symbolic features are combined to lead to better classification results.

Another example is the work of (Paiement, 2008) where accurate symbolic music models could dramatically improve the performance of transcription algorithms applied in more general contexts. They would provide "musical knowledge" to algorithms that currently only rely on basic sound properties to take decisions. In the same way, natural language models are commonly used in speech transcription algorithms (Rabiner and Schafer, 1978). As a simple example, suppose that a transcription algorithm knows the key of a particular song and tries to guess the last note of a song. The prior probability that this note would be the tonic would be very high, since most of the songs in any corpus end on the tonic.

Another advantage of symbolic music data is that it is much more compressed than audio data. For instance, the symbolic representation of an audio file of dozens of megabytes can be just a few kilobytes large. These few kilobytes contain most of the information that is needed to reconstruct the original audio file. Thus, we can concentrate on essential psychoacoustic features of the signal when designing algorithms to capture long term dependencies in symbolic music data. Finally, the most interesting advantage of dealing directly with symbolic data is the possibility of designing realistic music generation algorithms. Most of the probabilistic models presented in this thesis are generative models. Thus, these models can be sampled to generate genuine musical events, given other musical components or not. However, the generation of new music is out of the scope of this thesis.

1.2 The Melodic and Rhythmic Dimensions

Depending on the application domain, musical content is said to contain different attributes. In the psychoacoustics domain, (Levitin, 1999) uses pitch, rhythm, tempo, contour, timbre, loudness, and spatial location. In the MIR domain, (Downie, 1999) considers seven facets: pitch, temporal, harmonic, timbral, editorial, textural, and bibliographic. Other authors also include more elaborated features like thematic information obtained from the raw data (Hsu et al., 1998). For comparison based on musical content, the most used and directly available properties are pitch and rhythm. In this dissertation those two attributes will be used to describe musical notes, therefore the reader should known its definition.

1.2.1 Pitch

Pitch is the perceived frequency of a sound (Krumhansl, 1979). The frequency content of an idealized musical note is composed of a fundamental

frequency and integer multiples of that frequency. Human pitch perception is logarithmic with respect to fundamental frequency. Thus, we normally refer to the pitch of a note using pitch classes. In English, a pitch class is defined by a letter. For instance, the note with the fundamental frequency of 440 Hz is called A. In the Western music culture, the chromatic scale is the most common method of organizing notes. When using the equal temperament, each successive note is separated by a semitone. Two notes separated by a semitone have a fundamental frequency ratio of $2^{1/12}$ (approximately 1.05946). Using this system, the fundamental frequency is doubled every 12 semitones. The interval between two notes refers to the space between these two notes with regard to pitch. Two notes separated by 12 semitones are said to be separated by an octave, and have the same pitch-class. For instance, the note with fundamental frequency at 880 Hz is called A, one octave higher than the note A with the fundamental frequency at 440 Hz. We say that the interval between these two notes is an octave. Note that the note corresponding to 440 Hz is A_4 and A_5 corresponds to 880 Hz while its pitch class are the same, in this case A. The symbol '♯' (sharp) raises a note by one semitone. Conversely, the symbol '♭' (flat) lowers a note by one semitone. Most of the pitch classes are separated by one tone (i.e. two semitones), except for notes E and F, as well as B and C, that are separated only by one semitone.

In this system, $A^{\sharp}$ and $B^{\flat}$ refer to the same note. Two pitch classes that refer to the same pitch are called enharmonics. In this thesis, we consider enharmonics to be completely equivalent.

1.2.2 Rhythm

In most music notations, rhythm is defined relatively to an underlying beat that divides time in equal parts. The speed of the beat is called the tempo. For instance, when the tempo is 120, we count 120 beats per minute (BPM), or two beats per second. Meter is the sense of strong and weak beats that arises from the interaction among hierarchical levels of sequences having nested periodic components. Such a hierarchy is implied in Western music notation, where different levels are indicated by kinds of notes (whole notes, half notes, quarter notes, etc.) and where bars (or alternatively measures) establish segments containing an equal number of beats (Handel, 1989). Kinds of notes are defined relatively to each other. Whole notes have always twice the length of half notes, which have twice the length of quarter notes, and so on. The number of beats per bar is usually defined in the beginning of a song by the time signature. Also, depending on the meter definition, kinds of notes can last for variable number of beats. For instance, in most

four-beat meters, a quarter note lasts one beat. Hence, an eight note lasts for half a beat, a half note lasts for two beats, and a whole note lasts for four beats. If the tempo is 120, we play one half note per second and there is two half notes per bar.

1.3 Tree Representation

A number of works have addressed the problem of how to represent symbolic music in such a way that the comparison task can be both effective and efficient. Effectiveness can be measured in terms of their ability to cope with different aspects of human perception of melodic similarity. This is a difficult task and it mainly relies on both what are the data contained in the training sets of the carried out experiments and the comparison algorithms. On the other hand, efficiency is mainly related to time and memory complexities. This aspect assess to what extent the considered approach can be scaled to real world scenarios with hundreds or even thousands of music data.

Music melodies have been compared by different algorithms that use a variety of representations in the literature. String codings along with edit and alignment distances on a number of pitch and rhythm representations were used in (Grachten et al., 2005; Lemström, 2000; Mongeau and Sankoff, 1990) and n-gram model based algorithms were used in (Doraisamy, 2004; Downie, 1999; Uitdenbogerd, 2002). Also, a graph encoding was proposed in (Pinto and Tagliolato, 2008) in which the rhythm information was not represented. Other approaches for representation are less abstract and try to map melody pitches and durations into 2D plots, in a sort of piano roll, turning the melody matching problem into a geometric one (Aloupis et al., 2006; Tanur, 2005; Typke, 2007; Ukkonen et al., 2003; Wiggins et al., 2002).

The tree structure is an alternative representation between strings and graphs. On the one hand, its expressive capacity is higher than the strings and allow describing naturally hierarchical structures in which relations between its components are given. On the other hand, its management from the theoretical point of view is much simpler and efficient than that of graphs. Trees have been used by a number of authors in the literature with different aims. Formal language theory uses trees in a natural way and Lee (Lee, 1985) tried to interpret rhythms by using grammars. Something similar did Bod (Bod, 2002), but aiming to learn how to automatically segment melodies, using the tree approach provided by parsing the melody. In the work of Conklin (Gilbert and Conklin, 2007), monodies were parsed into tree structures using a probabilistic context-free grammar to perform melodic reductions that were used also in a segmentation task.

In the context of assisted musical composition, trees have been used as a way to conceptually represent music (Balaban, 1996; Smaill et al., 1993). Under this approach, the *Wind in the Willows* system (Högberg, 2005) used tree transducers to generate music. The renowned tool for assisted composition OpenMusic (Assayag et al., 1999) uses trees as a natural way for representing the hierarchical nature of duration subdivision of musical figures and groupings like tuplets.

For representing musicological analyses, the tree representation was used in the Generative Theory of Tonal Music (GTTM) (Lerdahl and Jackendoff, 1983) and in a number of works based on the Schenkerian analysis (Kirlin and Utgoff, 2008; Marsden, 2001, 2005, 2007; Smoliar, 1979).

Finally, trees have been also used not as a means to represent music, but as an intermediate data structure for other goals like building document structures for indexing (Blackburn, 2000; Drewes and Högberg, 2007; Skalak et al., 2008).

In this work, a tree encoding of melodies introduced in (Rizo, 2010; Rizo et al., 2003) has been employed that uses the tree structure to encode rhythm implicitly. In this structure, leaves contain pitches. Duration is represented by the level of the tree: the shorter a note is, the deeper it appears in the tree and an in-depth tree traversal represents the time sequence of music. This representation has proven to be effective in a number of melodic similarity computation in tasks (Habrard et al., 2008; Rizo, 2010). In Chapter 3 this encoding is described in more detail.

One of the main advantages of melodic trees is that, unlike linear representations in which both melodic dimensions: time (duration) and pitch are coded by explicit symbols, they implicitly represent time in their structure, making use of the fact that note durations are multiples of basic time units in a binary (sometimes ternary, depending on meter and some figures like triplets, etc.). This way, they are less sensitive to melody coding issues, since only pitch codes are needed to be established, so there are less degrees of freedom for coding and, therefore, less parameters to be tuned.

However, this representation has some main drawbacks: its tight dependency on the meter structure of the input source, and its difficulty to represent ties, dots, and syncopations. The first problem can be overcome through an a priori metrical analysis of the work (Eck and Casagrande, 2005; Meudic, 2002) in the case that the meter metadata is not present in the data source. The second drawback, from the representation point of view, can be solved by the addition of a special symbol that encodes the concept of note continuation. For the comparison task, this is not a problem as other authors point out (Hanna et al., 2008; Mongeau and Sankoff, 1990; Pardo and Sanghi, 2005). One derived problem is the excessive growth of trees

when very short notes and performance imprecisions are found in the case of real time sequenced data. This problem has been addressed by tree pruning methods and advanced quantization algorithms (Agon et al., 1994; Cemgil et al., 2000). Finally, as we explain in the next section, the methods to compare this kind of structures are costly, in some cases become almost intractable and more efficient methods are needed.

1.4 Objectives and the Approach in this Thesis

Due to the advantages of the tree structure, described previously, our motivation to do this dissertation is trying to built a system that allowed to classify and generate melodies using the information from the tree encoding, capturing the inherent dependencies which are inside this kind of structure, and improving the current methods in terms of accuracy and running time. In this way, we are trying to find more efficient methods that is key to use the tree structure in large datasets.

First, we study the possibilities of the tree edit similarity to classify melodies using a new approach for estimate the weights of the edit operations. Focusing on tree similarity learning problem, we investigate a new framework for learning tree edit distances thanks to a convex optimization problem based on the framework called *GESL* for *Good Edit Similarity Learning* and described in (Bellet et al., 2012), originally developed for strings.

Once the possibilities of the cited approach are studied, an alternative approach is used. For that a grammatical inference approach is used to infer tree languages. The inference of these languages give us the possibility to use them to classify new trees (melodies). Moreover, this approach could be used to generate new samples even though this question is out of the scope of this thesis.

In order to carry out this research we focus in two different approaches. On the one hand, the formalism of tree automata is used to infer the models and classify melodies represented as trees. For this, the k-testables tree languages will be used. On the other hand, tree grammars are inferred from the tree automata. In this way, the grammars inferred using tree data in training are used to classify new melodies represented as strings. As we explain throughout this dissertation the last approach is needed to solve the problem when the duration information is not available in the samples that have to be classified.

1.4.1 Machine Learning.

The tasks cited above will be faced in the context of methods able to learn or infer the models from data examples. The field that deals with studying this kind of systems is called machine learning.

Machine learning is a subfield of artificial intelligence which seeks to answer the scientific question of how we can build computer systems that are able to learn. Analysing these systems and trying to find general laws which govern learning processes is also of interest to the field.

The utility and applications of such computer systems are vast. Such systems are already being used nowadays in a variety of areas such as natural language processing, bioinformatics, robotics, stock market analysis and others.

The goal of machine learning is to automatically figure out how to perform tasks by generalizing from examples. A machine learning algorithm takes a data sample as input and infers a model that captures the underlying mechanism (usually assumed to be some unknown probability distribution) which generated the data. Data can consist of features vectors (e.g., the age, body mass index, blood pressure, ... of a patient) or can be structured, such as strings (e.g., text documents) or trees (e.g., XML documents). A classic setting is supervised learning, where the algorithm has access to a set of training examples along with their labels and must learn a model that is able to accurately predict the label of future (unseen) examples. Supervised learning encompasses classification problems, where the label set is finite (for instance, predicting the label of a character in a handwriting recognition system) and regression problems, where the label set is continuous (for example, the temperature in weather forecasting). On the other hand, an unsupervised learning algorithm has no access to the labels of the training data. A classic example is clustering, where we aim at assigning data into similar groups. The generalization ability of the learned model (i.e., its performance on unseen examples) can sometimes be guaranteed using arguments from statistical learning theory. Relying on the saying birds of a feather flock together, many supervised and unsupervised machine learning algorithms are based on a notion of metric (similarity or distance function) between examples, such as k-nearest neighbors or support vector machines in the supervised setting and K-Means clustering in unsupervised learning. The performance of these algorithms critically depends on the relevance of the metric to the problem at hand for instance, we hope that it identifies as similar the examples that share the same underlying label and as dissimilar those of different labels. Unfortunately, standard metrics (such as the Euclidean distance between feature vectors or the edit distance between

strings) are often not appropriate because they fail to capture the specific nature of the problem of interest.

For this reason, a lot of effort has gone into metric learning, the research topic devoted to automatically learning metrics from data. In this thesis, we focus on supervised metric learning, where we try to adapt the metric to the problem at hand using the information brought by a sample of labeled examples. Many of these methods aim to find the parameters of a metric so that it best satisfies a set of local constraints over the training sample, requiring for instance that pairs of examples of the same class should be similar and that those of different class should be dissimilar according to the learned metric. A large body of work has been devoted to supervised metric learning from feature vectors, in particular Mahalanobis distance learning, which essentially learns a linear projection of the data into a new space where the local constraints are better satisfied. While early methods were costly and could not be applied to medium-sized problems, recent methods offer better scalability and interesting features such as sparsity. Supervised metric learning from structured data has received less attention because it requires more complex procedures. Most of the work has focused on learning metrics based on the edit distance. Roughly speaking, the edit distance between two objects corresponds to the cheapest sequence of edit operations (insertion, deletion and substitution of subparts) turning one object into the other, where operations are assigned specific costs gathered in a matrix. Edit distance learning consists in optimizing the cost matrix and usually relies on maximizing the likelihood of pairs of similar examples in a probabilistic model.

1.4.2 Improving the Similarity. Tree Edit Similarity Learning.

Searching in the literature we can find many applications require to deal with hierarchical information represented as trees such as Web/XML data processing in web information retrieval, syntactic analysis in natural language processing, structured databases, structured representations in images or symbolic representations in music information retrieval. In these areas, many pattern recognition algorithms are used to perform classification or clustering tasks. A lot of them rely on a notion of distance or similarity such as k-Nearest-Neighbors, k-means or SVM-based classifiers. In the context of tree-structured data, tree edit distance (TED) (Bille, 2005) is admitted to be a relevant measure. TED is defined as the least costly set of basic operations to transform one tree into another. These basic operations are

based on the substitution, insertion or deletion of nodes. There are two main variants of the TED differing in the way the insertion and deletion of the nodes are handled. In the Selkow variant, insertion and deletion operations are restricted to the leaves of the tree or, in other words, these two operations can only be applied to full subtrees. In the other variant, deletion or insertion operations can be applied directly to internal nodes, the corresponding distance can be evaluated by the Zhang and Shasha algorithm (Zhang and Shasha., 1989) for example. TED-based approaches generally use *a priori* fixed costs for these edit operations. However, using a distance tailored to a specific task is key to the success of many algorithms.

Following this idea, some research effort has been devoted into learning the edit costs from a learning data. The first existing approach (Neuhaus and Bunke, 2004) proposes to learn a (more general) graph edit distance, where each edit operation is modeled by a Gaussian mixture density whose parameters are learned using an EM-like algorithm. In (Zigoris et al., 2006), the authors make use of a variant of SVM to learn a parameterized tree alignment model for extracting fields from HTML search results. Their learning method is based on an optimization problem solved thanks to very time-consuming EM-like algorithm which becomes intractable when the trees become large. Another strategy has been proposed in (Bernard et al., 2008) by learning a stochastic version of the tree edit distance under the form of a stochastic edit transducer. The associated learning algorithm is also an EM-based approach tailored to the Selkow variant. This method has been extended to the Zhang and Shasha algorithm in (Boyer et al., 2007). The two previous approaches are limited by the fact that they model distribution on tree edit scripts rather than on trees themselves (unlike the case of strings). The method presented in (Dalvi et al., 2009) has proposed a solution to overcome this problem by using a more complex conditional transducer using again an EM-like stratergy to infer the parameters. The learning algorithm used in (Mehdad, 2009) proposes a rather different method based on the Particle Swarm Optimization. The edit costs are learned via an iterative process which may not induce a true metric and is not provably effective. Recently, the approach (Emms, 2012) points out a theoretical limitation of (Boyer et al., 2007) coming from the fact the factorization used in this paper can turns out to be incorrect in some cases. The author provides a correct factorization when using the Vitervi edit script (*i.e.* the tree edit script of highest probability). However, this approach still suffers from many drawbacks including overfitting.

From this short survey, we can note that few methods have been proposed in the literature. The most striking limitation is essentially due to algorithmic

constraints coming from the underlying complexity of evaluating the tree edit distance. Indeed, computation of tree edit distance is generally costly: For example, the Zhang and Shasha algorithm requires cubic time while Selkow's variant involves a quadratic time complexity. This high complexity is increased when the learning procedure is based on the EM algorithm due to the iterations this procedure requires. In this context, tree distance learning becomes almost intractable and more efficient methods are needed.

Following these works, we propose to consider a new framework for learning edit similarities addressing the main drawbacks described above. This approach, called *GESL* for *Good Edit Similarity Learning* and described in (Bellet et al., 2012), is tailored to binary classification, originally designed for string-structured data. It proposes to optimize the costs of the edit operations involved in the classical string edit distance (also called the Levenshtein distance (Levenshtein, 1966)) to fulfill a criterion of good similarity according to the theory of (ϵ, γ, τ)-good similarity functions presented in (Balcan et al., 2008). This theory bridges the gap between the property of a good similarity function and its performance in linear classification and thus its discriminative power. GESL has shown that this framework is well suited to edit similarity learning. Indeed, it provides a procedure having three main advantages. First, the learning algorithm is very efficient: The Levenshtein edit scripts need to be computed only once and the edit costs are then assessed by solving an optimization program defined as convex quadratic program that can be very efficiently solved. Second, GESL has strong theoretical guarantees in the form of a generalization bound which is independent from the size of the alphabet making the approach suitable for problems with high alphabet. Finally, this generalization bound justifies that this similarity can be used to learn low-error classifiers for the task at hand. This approach seems thus very appropriate to learn powerful edit similarity. However, this approach has never been applied in the context of tree-structured data.

We provide two contributions about this. First, we are trying to prove experimentally that the methodology of GESL can be successfully used with tree-structured data. We also are providing evidence that this method can be easily used with various distances by considering both Selkow and Zhang-Shasha variants. On the other hand, we are trying to show that this type of model is very interesting for music information retrieval tasks. Indeed, the edit measure directly works on the structure of the data and thus on the melody structure itself. Edit operations used to discriminate some musical pieces can then provide a valuable understandable knowledge. For example,

in music analysis, one can wonder what are the most common changes among the possible variations of musical tunes that make the tune still recognizable. Our hypothesis is that those changes must have small edit costs, in opposition to high valued edit operations which then may correspond to information used to discriminate some pieces. We will show in our experiments how this knowledge can be interpreted.

1.4.3 Similarity as a Probability to Belong to a Tree Language.

However, GESL has some drawbacks: on the one hand, if new samples are collected to train the models, GESL needs to train the costs with all the samples including the new ones, consuming in this way an important quantity of time. On the other hand, GESL is to compare trees by trees or strings by strings but if we want classify an string using the information inside the tree structure, it is not possible.

To try to solve these problems we propose use another way to compare trees. This is using grammatical inference to learning a tree language. Then the probability of a tree (melody) to belong to a language can be calculated and use this to classify the melody. Language modeling has been also selected for the experiments in this thesis because it allows to study the paralelism between music and natural language, studying the ability of language models built from a training corpus to predict the classification of new songs. Moreover, these models could be used as a generative model to create new genuine musical events.

Grammatical Inference One branch of machine learning is Grammatical Inference (GI). In GI the problem is to learn formal language representations from information about an unknown target formal language. Formal languages are simply (possibly infinite) sets of strings and a representation of a formal language L is any finite model which is able to generate or exactly describe L. Typical formal language representations take the form of grammars or automata. Therefore, the hypothesis space in GI is most often some class of grammars or automata (e.g. regular automata or context-free grammars) which contains all the possible models that can be learned. GI can be applied wherever grammars or automata serve as good models for some task. For example, in Natural Language Processing (NLP), Probabilistic Context-Free Grammars (PCFGs) serve as good models for assigning probabilities to sentences and describing the structure of sentences

(where these tasks can prove very useful for speech recognition, machine translation and other NLP related tasks). Thus, GI can be used to induce such grammars from corpora of unstructured raw natural language sentences. Another example is Bioinformatics, where Hidden Markov Models (HMMs) and PCFGs have proven to be good descriptors for DNA, RNA and protein structures. Practical GI learning algorithms that infer finite state automata include (Lang et al., 1998) (Adriaans and Jacobs, 2006) (Coste and Nicolas, 1998) (Heule and Verwer, 2010), and for the context-free case ((Sakakibara et al., 1994) ; (Nakamura and Ishiwata, 2000) (Stolcke, 1994) (Petasis et al., 2004) (Adriaans et al., 2000) (Zaanen, 2001) (Clark, 2001).

The reader is referred to (de la Higuera, 2010) for a comprehensive review of the grammatical inference field.

Our contribution in this subject is prove that grammatical inference is suitable to learn a tree language from music represented as trees. Then, these models can be used to classify new music. First, to learn a language using tree data we use probabilistic k-testable tree models (Rico-Juan et al., 2005), a generalization of the k-gram models. These models are easy to infer from samples and allow incremental updates. They can be used for data categorization if a model is inferred for each class and the new samples are assigned a probability by each model, taking a maximum likelihood decision. These models have been used in a number of applications, like structured data compression (Rico-Juan et al., 2005) or information extraction from structured documents (like HTML or XML, for example) (Kosalaa et al., 2006), but here they are applied to music data, focusing on the aforementioned inherent structured nature.

These models build a stochastic tree automata that parses a tree from the leaves (initial states) to its root (final states), assigning probabilities to the transitions. In this process, the similarity between a melody and a reference model is computed as a probability.

However, probabilistic k-testable tree models have a limitation. These models only work with tree data and therefore they does not work when the duration information is not available for the input data. We propose a solution when we have melodies represented by trees for the training but the duration information is not available for the input data. For that, we infer a probabilistic context-free grammar using the information in the trees (duration and pitch) and classify new melodies represented by strings using only the pitch. The case study in this work is to identify a snippet query among a set of songs stored in symbolic format. For it, the utilized method must be able to deal with inexact queries and efficient for scalability issues.

For it, we use probabilistic tree grammars (Verdu-Mas et al., 2005). These grammars can be obtained from probabilistic k-testable tree models (Rico-Juan et al., 2005). The duration information implicit in the tree representation is captured by the grammar and this is used for classifying the new melodies represented by strings that only have the pitch information. This is a solution when duration information is not available or unreliable for the input data.

Moreover, symbolic music retrieval from queries has been extensively studied in the MIR literature. In order to search a query in a dataset of songs, we can apply any of the well-known pattern matching algorithms like local string editing to each of the songs in the dataset, and retrieve a ranking of most similar items. The main problem here is the efficiency when using large datasets, but with the advantage that it can find a partial or approximate occurrence of the query in any part of the dataset. On the other hand, a previous indexation of the dataset, e.g by means of motive extraction, solves the scalability issue, but motive extraction usually needs to work with exact repetitions, making very inaccurate to build robust indices from different interpretations of the same song. In addition, when searching only in motives, the music not present in the motives is hidden.

Our proposal is able to solve intrinsically these problems. The probabilistic grammar structure itself encodes both motives and melody variations, giving more weight to the most repeated themes, without the need to be exact, and enabling the possibility to learn from different renderings of the same song, leaving aside the need to encode motives. Besides, there is no difference between looking for a whole song or searching a small query in the dataset.

1.5 Structure of this Thesis

The rest of this manuscript is structured as follows:

Chapter 2: Technical Background. In this chapter the supervised learning setting and analytical frameworks for deriving generalization guarantees are presented. Moreover, some metrics are described.

Chapter 3: Melody Tree Representation. Tree representation review and presentation of the tree structured used.

Chapter 4: Tree Similarity Learning. Learning tree edit similarities called GESL (for Good Edit Similarity Learning) that relies on a relaxed version of (ϵ, γ, τ)-goodness.

Chapter 5: Tree Automata. Presents the studies carried out for the Stochastic k-testable tree automata. In this chapter melodies represented as trees are classified using these models.

Chapter 6: Tree Grammars. Presents the studies carried out for the Stochastic k-testable tree grammars. The grammars inferred using tree data in training are used to classify new melodies represented as strings.

Chapter 7: Conclusions and Future Work Conclusions and contributions of this thesis are summarized here. Future research lines are outlined.

2
Technical Background

In this chapter the supervised learning setting and analytical frameworks for deriving generalization guarantees are presented. Moreover, some metrics are described. Both of them help us to understand the problems derived and the solutions proposed in this thesis.

2.1 Supervised Learning

The goal of supervised learning is to automatically infer a model (hypothesis) from a set of labeled examples that is able to make predictions given new unlabeled data. In the following, we review basic notions of statistical learning theory, a very popular framework pioneered by (Vapnik and Chervonenkis, 1971). The interested reader can refer to (Vapnik, 1998) and (Bousquet et al., 2003) for a more thorough description.

2.1.1 Typical Setting

In supervised learning, we learn a hypothesis from a set of labeled examples. This notion of training sample is formalized below.

Definition 1 *(Training sample). A training sample of size n is a set $T = \{z_i = (x_i, y_i)\}_{i=1}^{n}$ of n observations independently and identically distributed (i.i.d.) according to an unknown joint distribution P over the space $Z = X \times Y$, where X is the input space and Y the output space. For a given observation $z_i, x_i \in X$ is the instance (or example) and $y_i \in Y$ its label. When Y is discrete, we are dealing with a classification task, and y_i is called the class of x_i. When Y is continuous, this is a regression task.*

Definition 2 *(Alphabet and trees). An alphabet Σ is a finite nonempty set of symbols. Let Σ be a set of labels and \$ be the empty symbol, T_Σ being the*

set of all possible trees (including the empty tree \$). Let $t = \sigma(t_1, \ldots, t_n)$ be a tree where σ is a node and $t_1, \ldots, t_n$ is an ordered sequence of (sub)trees. σ is called the root of t. $t_1, \ldots, t_n$ are called the children of σ. A node with no child ($n = 0$) is a leaf. The size of the tree t is denoted by $|t|$ and corresponds to the number of nodes constituting the tree. In the following, we assume each node to be labeled by a symbol σ coming from a set of labels Σ. We denote by T_Σ the set of trees that can be built from Σ.

We can now formally define what we mean by supervised learning.

Definition 3 *(Supervised learning). Supervised learning is the task of inferring a function (often referred to as a hypothesis or a model) $h_T : X \to L$ belonging to some hypothesis class H from a training sample T , which best predicts y from x for any (x, y) drawn from P. Note that the decision space L may or may not be equal to Y.*

In order to choose h_T , we need a criterion to assess the quality of an arbitrary hypothesis h. Given a nonnegative loss function $l : H \times Z \to R^+$ measuring the degree of agreement between $h(x)$ and y, we define the notion of true risk.

Definition 4 *(True risk). The true risk (also called generalization error) $R^l(h)$ of a hypothesis h with respect to a loss function l is the expected loss suffered by h over the distribution P:*

$$R^l(h) = E_{z\sim P}[l(h, z)].$$

The most natural loss function for binary classification is the 0/1 loss (also called classification error):

$$l_{0/1}(h, z) = \begin{cases} 1 & \text{if } yh(x) < 0, \\ 0 & \text{otherwise.} \end{cases}$$

$R^{l_{0/1}}(h)$ then corresponds to the proportion of time $h(x)$ and y agree in sign, and in particular to the proportion of correct predictions when $L = Y$.

The goal of supervised learning is then to find a hypothesis that achieves the smallest true risk. Unfortunately, in general we cannot compute the true risk of a hypothesis since the distribution P is unknown. We can only measure it empirically on the training sample. This is called the empirical risk.

Definition 5 *(Empirical risk). Let* $T = \{z_i = (x_i, y_i)\}_{i=1}^n$ *be a training sample. The empirical risk (also called empirical error)* $R_T^l(h)$ *of a hypothesis* h *over* T *with respect to a loss function* l *is the average loss suffered by* h *on the instances in* T*:*

$$R_T^l(h) = \frac{1}{n} \sum_{i=0}^{n} l(h, z_i).$$

Under some restrictions, using the empirical risk to select the best hypothesis is a good strategy, as discussed in the next section.

2.1.2 Finding a Good Hypothesis

This section focuses on classic strategies for finding a good hypothesis in the true risk sense. The derivation of guarantees on the true risk of the selected hypothesis will be studied in Section 2.2.

Simply minimizing the empirical risk over all possible hypotheses would obviously be a good strategy if infinitely many training instances were available. Unfortunately, in realistic scenarios, training data is limited and there always exists a hypothesis h, however complex, that perfectly predicts the training sample, i.e., $R_T^l(h) = 0$, but generalizes poorly, i.e., h has a nonzero (potentially large) true risk. This situation where the true risk of a hypothesis is much larger than its empirical risk is called overfitting. The intuitive idea behind it is that learning the training sample by heart does not provide good generalization to unseen data.

There is therefore a trade-off between minimizing the empirical risk and the complexity of the considered hypotheses, known as the bias-variance trade-off. There essentially exist two ways to deal with it and avoid overfitting: (i) restrict the hypothesis space, and (ii) favor simple hypotheses over complex ones. In the following, we briefly present three classic strategies for finding a hypothesis with small true risk.

Empirical Risk Minimization The idea of the Empirical Risk Minimization (ERM) principle is to pick a restricted hypothesis space $H \subset L^X$ (for instance, linear classifiers, decision trees, etc.) and select a hypothesis $h_T \in H$ that minimizes the empirical risk:

$$h_T = \arg\min_{h \in H} R_T^l(h).$$

This may work well in practice but depends on the choice of hypothesis space. Essentially, we want H large enough to include hypotheses with small risk, but H small enough to avoid overfitting. Without background knowledge on the task, picking an appropriate H is difficult.

Structural Risk Minimization In Structural Risk Minimization (SRM), we use an infinite sequence of hypothesis classes $H_1 \subset H_2 \subset \ldots$ of increasing size and select the hypothesis that minimizes a penalized version of the empirical risk that favors simple classes:

$$h_T = \arg \min_{h \in H_c, c \in N} R^l_T(h) + pen(H_c).$$

This implements the Occam's razor principle according to which one should choose the simplest explanation consistent with the training data.

Regularized Risk Minimization Regularized Risk Minimization (RRM) also builds upon the Occam's razor principle but is easier to implement: one picks a single, large hypothesis space H and a regularizer (usually some norm $||h||$) and selects a hypothesis that achieves the best trade-off between empirical risk minimization and regularization:

$$h_T = \arg \min_{h \in H} R^l_T(h) + \lambda ||h||, \tag{2.1}$$

where λ is the trade-off parameter (in practice, it is set using validation data). The role of regularization is to penalize complex hypotheses. Note that it also provides a built-in way to break the tie between hypotheses that have the same empirical risk.

The choice of regularizer is important and depends on the considered task and the desired effect. Some regularizers are easy to optimize because they are convex and smooth (for instance, the squared L_2 norm) while others do not have these convenient properties and are thus harder to deal with. However, the latter may bring some potentially interesting effects such as sparsity: they tend to set some parameters of the hypothesis to zero.

Regularization is used in many successful learning methods and, as we will see in Section 2.2, may help deriving generalization guarantees.

2.1.3 Surrogate Loss Functions

The methods described above all rely on minimizing the empirical risk. However, due to the nonconvexity of the 0/1 loss, minimizing (or approximately minimizing) $R^{l_{0/1}}$ is known to be NP-hard even for simple hypothesis classes

(Ben-David et al., 2003). For this reason, surrogate convex loss functions (that can be more efficiently handled) are often used. The most prominent choices in the context of binary classification are:

- the hinge loss: $l_{hinge}(h, z) = [1 - yh(x)]_+ = \max(0, 1 - yh(x))$, used for instance in support vector machines (Cortes and Vapnik, 1995).
- the exponential loss: $l_{exp}(h, z) = e^{-yh(x)}$, used in Adaboost (Freund and Schapire, 1995).
- the logistic loss: $l_{log}(h, z) = \log(1 + \epsilon^{-yh(x)})$,used in Logitboost (Friedman et al., 2000).

Choosing an appropriate loss function is not an easy task and strongly depends on the problem, but there exist general results on the relative merits of different loss functions. For instance, (Rosasco et al., 2004) studied statistical properties of several convex loss functions in a general classification setting and concluded that the hinge loss has a better convergence rate than other loss functions. (Ben-David et al., 2012) have further shown that in the context of linear classification, the hinge loss offers the best guarantees in terms of classification error. In the following section, we present analytical frameworks that allow the derivation of generalization guarantees, i.e., relating the empirical risk of h_T to its true risk.

2.2 Deriving Generalization Guarantees

In the previous section, we described a few generic methods for learning a hypothesis h_T from a training sample T based on minimizing the (penalized) empirical risk. However, learning a hypothesis with small true risk is what we are really interested in. Typically, the empirical risk can be seen as an optimistically biased estimation of the true risk (especially when the training sample is small), and a considerable amount of research has gone into deriving generalization guarantees for learning algorithms, i.e., bounding the deviation of the true risk of the learned hypothesis from its empirical measurement. These bounds are often referred to as PAC (Probably Approximately Correct) bounds (Valiant, 1984) and have the following form:

$$P_r[|R^l(h) - R^l_T(h)| > \epsilon] \leq \delta,$$

where $\epsilon \geq 0$ and $\delta \in [0, 1]$. In other words, it bounds the probability

to observe a large gap between the true risk and the empirical risk of an hypothesis.

The key instruments for deriving PAC bounds are concentration inequalities. They essentially assess the deviation of some functions of independent random variables from their expectation. Different concentration inequalities tackle different functions of the variables. The most commonly used in machine learning are Chebyshev (only one variable is considered), Hoeffding (sums of variables) and McDiarmid (that can accommodate any sufficiently regular function of the variables). For more details about concentration inequalities, see for instance the survey of (Boucheron et al., 2004).

In this section, we present two theoretical frameworks for establishing generalization bounds: uniform convergence and uniform stability (for a more general overview, please refer to the tutorial by (Langford, 2005). Note that the approach used in our contributions in Chapter 4 make use of these frameworks.

2.2.1 Uniform Convergence

The theory of uniform convergence of empirical quantities to their mean (Vapnik and Chervonenkis, 1971) (Vapnik, 1982) is one of the most prominent tools for deriving generalization bounds. It provides guarantees that hold for any hypothesis $h \in H$ (including h_T) and essentially bounds (with some probability $1-\delta$) the true risk of h by its empirical risk plus a penalty term that depends on the number of training examples n, the size (or complexity) of the hypothesis space H and the value of δ. Intuitively, large n brings high confidence (since as $n \to \infty$ the empirical risk converges to the true risk by the law of large numbers), complex H brings low confidence (since overfitting is more likely), and δ accounts for the probability of drawing an unlucky training sample (i.e., not representative of the underlying distribution P).

When the hypothesis space is finite, we get the following PAC bound in $O(1/\sqrt{n})$.

Theorem 1 *(Uniform convergence bound for the finite case). Let T be a training sample of size n drawn i.i.d. from some distribution P, H a finite hypothesis space and $\delta > 0$. For any $h \in H$, with probability $1-\delta$ over the random sample T, we have:*

$$R^l(h) \leq R^l_T(h) + \sqrt{\frac{ln|H| + ln(1/\delta)}{2n}}.$$

When H is continuous (for instance, if H is the space of linear classifiers), we need a measure of the complexity of H such as the VC dimension (Vapnik

and Chervonenkis, 1971), the fat-shattering dimension (Alon et al., 1997) or the Rademacher complexity (Koltchinskii, 2001) (Bartlett and Mendelson, 2002). For instance, using the VC dimension, we get the following bound.

Theorem 2 *(Uniform convergence bound with VC dimension). Let T be a training sample of size n drawn i.i.d. from some distribution P, H a continuous hypothesis space with VC dimension $VC(H)$ and $\delta > 0$. For any $h \in H$, with probability $1 - \delta$ over the random sample T , we have:*

$$R^l(h) \leq R^l_T(h) + \sqrt{\frac{VC(H)(ln\frac{2n}{VC(H)} + 1) + ln(4/\delta)}{n}}.$$

A drawback of uniform convergence analysis is that it is only based on the size of the training sample and the complexity of the hypothesis space, and completely ignores the learning algorithm, i.e., how the hypothesis h_T is selected. In the following, we present an analytical framework that explicitly take into account the algorithm and can be used to derive generalization guarantees for h_T specifically, in particular in the regularized risk minimization setting 2.1.

2.2.2 Uniform Stability

Building on previous work on algorithmic stability, (Bousquet and Elisseeff, 2001) (Bousquet and Elisseeff, 2002) introduced new definitions that allow the derivation of generalization bounds for a large class of algorithms. Intuitively, an algorithm is said stable if it is robust to small changes in its input (in our case, the training sample), i.e., the variation in its output is small. Formally, we focus on uniform stability, a version of stability that allows the derivation of rather tight bounds.

Definition 6 *(Uniform stability). An algorithm A has uniform stability k/n with respect to a loss function l if the following holds:*

$$\forall T, |T| = n, \forall i \in [n] : \sup_z |l(h_T, z) - l(h_{T^i}, z)| \leq \frac{k}{n},$$

where k is a positive constant, T^i is obtained from the training sample T by replacing the i^{th} example $z_i \in T$ by another example z'_i drawn i.i.d. from P , h_T and h_{T^i} are the hypotheses learned by A from T and T^i respectively.

(Bousquet and Elisseeff, 2001) (Bousquet and Elisseeff, 2002) have shown that a large class of regularized risk minimization algorithms satisfies this

definition. The constant k typically depends on the form of the loss function, the regularizer and the regularization parameter λ. Making a good use of McDiarmid's inequality, they show that when Definition 6 is fulfilled, the following bound in $O(1/\sqrt{n})$ holds.

Theorem 3 *(Uniform stability bound). Let T be a training sample of size n drawn i.i.d. from some distribution P and $\delta > 0$. For any algorithm A with uniform stability k/n with respect to a loss function l upper-bounded by some constant B, with probability $1 - \delta$ over the random sample T, we have:*

$$R^l(h_T) \leq R^l_T(h_T) + \frac{k}{n} + (2k + B)\sqrt{\frac{ln(1/\delta)}{2n}},$$

where h_T is the hypothesis learned by A from T.

The main difference between uniform convergence and uniform stability is that the latter incorporates regularization (through k and h_T) and does not require any hypothesis space complexity argument. In particular, uniform stability can be used to derive generalization guarantees for hypothesis classes that are difficult to analyze with classic complexity arguments, such as k-nearest neighbors or support vector machines that have infinite VC dimension. It can also be adapted to non-i.i.d. settings (Mohri and Rostamizadeh, 2007) (Mohri and Rostamizadeh, 2010). We will use uniform stability in the contributions presented in Chapter 4.

2.3 Metrics

After having presented the supervised learning setting and analytical frameworks for deriving generalization guarantees, we now turn to the topic of metrics, which has a great place in this thesis.

The notion of metric (used here as a generic term for distance, similarity or dissimilarity function) plays an important role in many machine learning problems such as classification, regression, clustering, or ranking. Successful examples include:

- k-Nearest Neighbors (k-NN) classification (Cover and Hart, 2006), where the predicted class of an instance x corresponds to the majority class among the k-nearest neighbors of x in the training sample, according to some distance or similarity.

- Kernel methods (Scholkopf and Smola, 2001), where a specific type of similarity function called kernel (see Definition 9) is used to implicitly project data into a new high-dimensional feature space. The most prominent example is Support Vector Machines (SVM) classification (Cortes and Vapnik, 1995), where a large-margin linear classifier is learned in that space.

- K-Means (Lloyd, 2006), a clustering algorithm which aims at finding the K clusters that minimize the within-cluster distance on the training sample according to some metric.

- Information retrieval, where a similarity function is often used to retrieve documents (webpages, images, etc.) that are similar to a query or to another document (Salton et al., 1975) (Baeza-Yates and Ribeiro-Neto, 1999) (Sivic and Zisserman, 2009).

- Data visualization, where visualization of interesting patterns in high-dimensional data is sometimes achieved by means of a metric (Venna et al., 2010) (Bertini et al., 2011).

It should be noted that metrics are especially important when dealing with structured data (such as strings, trees, or graphs) because they are often a convenient proxy to manipulate these complex objects: if a metric is available, then any metric-based algorithm (such as those presented in the above list) can be used.

In this section, we first give the definitions of distance, similarity and kernel functions 2.3.1, and then give some examples (by no means an exhaustive list) of such metrics between structured data 2.3.2.

2.3.1 Definitions

We start by introducing the definition of a distance function.

Definition 7 *(Distance function). A distance over a set X is a pairwise function $d : X \times X \to R$ which satisfies the following properties $\forall x, x', x'' \in X$:*

1. $d(x, x') \geq 0$ *(nonnegativity),*
2. $d(x, x') = 0$ *if and only if* $x = x'$ *(identity of indiscernibles),*
3. $d(x, x') = d(x', x)$ *(symmetry),*

4. $d(x, x'') \leq d(x, x') + d(x', x'')$ *(triangle inequality).*

A pseudo-distance satisfies the properties of a metric, except that instead of property 2, only $d(x, x) = 0$ is required. Note that the property of triangle inequality can be used to speedup learning algorithms such as k-NN (e.g., (Micó et al., 1994), (Lai et al., 2007), (Wang, 2011)) or K-Means (Elkan, 2003).

While a distance function is a well-defined mathematical concept, there is no general agreement on the definition of a (dis)similarity function, which can essentially be any pairwise function. Throughout this thesis, we will use the following definition.

Definition 8 *(Similarity function). A (dis)similarity function is a pairwise function $K : X \times X \to [-1, 1]$. We say that K is a symmetric similarity function if $\forall x, x' \in X, K(x, x') = K(x', x)$.*

A similarity function should return a high score for similar inputs and a low score for dissimilar ones (the other way around for a dissimilarity function). Note that (normalized) distance functions are dissimilarity functions.

Finally, a kernel is a special type of similarity function, as formalized by the following definition.

Definition 9 *(Kernel function). A symmetric similarity function K is a kernel if there exists a (possibly implicit) mapping function $\phi : X \to H$ from the instance space X to a Hilbert space H such that K can be written as an inner product in H:*

$$K(x, x') = \langle \phi(x), \phi(x') \rangle.$$

Equivalently, K is a kernel if it is positive semi-definite (PSD), i.e.,

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j K(x_i, x_j) \geq 0$$

for all finite sequences of $x_1, \ldots, x_n \in X$ and $c_1, \ldots, c_n \in R$.

Kernel functions are a key component of kernel methods such as SVM, because they can implicitly allow cheap inner product computations in very high-dimensional spaces (this is known as the "kernel trick") and bring an elegant theory based on Reproducing Kernel Hilbert Spaces (RKHS). Note that these advantages disappear when using an arbitrary non-PSD similarity function instead of a kernel, and the convergence of the kernel-based algorithm may not even be guaranteed in this case.

2.3.2 Some Metrics between Structured Data

Hamming distance The Hamming distance is a distance between strings of identical length and is equal to the number of positions at which the symbols differ. It has been used mostly for binary strings and is defined by

$$d_{ham}(x, x') = |\{i : x_i \neq x'_i\}|.$$

String edit distance The string edit distance (Levenshtein, 1966) is a distance between strings of possibly different length built from an alphabet Σ. It is based on three elementary edit operations: insertion, deletion and substitution of a symbol. In the more general version, each operation has a specific cost, gathered in a nonnegative $(|\Sigma| + 1) \times (|\Sigma| + 1)$ matrix C (the additional row and column account for insertion and deletion costs respectively). A sequence of operations transforming a string x into a string x' is called an edit script. The edit distance between x and x' is defined as the cost of the cheapest edit script that turns x into x' and can be computed in $O(|x| \cdot |x'|)$ time by dynamic programming.

The classic edit distance, known as the Levenshtein distance, uses a unit cost matrix and thus corresponds to the minimum number of operations turning one string into another. For instance, the Levenshtein distance between *abb* and *aa* is equal to 2, since turning *abb* into *aa* requires at least 2 operations (e.g., substitution of *b* with a and deletion of *b*).

Using task-specific costs is a key ingredient to the success of the edit distance in many applications. For some problems such as handwritten character recognition (Micó and Oncina, 1998) or protein alignment (Dayhoff et al., 1978), (Henikoff and Henikoff, 1992), relevant cost matrices may be available. But a more general solution consists in automatically learning the cost matrix from data.

Sequence alignment Sequence alignment is a way of computing the similarity between two strings, mostly used in bioinformatics to identify regions of similarity in DNA or protein sequences (Mount, 2004). It corresponds to the score of the best alignment. The score of an alignment is based on the same elementary operations as the edit distance and on a score matrix for substitutions, but uses a (linear or affine) gap penalty function instead of insertion and deletion costs. The most prominent sequence alignment measures are the Needleman-Wunsch score (Needleman and Wunsch, 1970) for global alignments and the Smith-Waterman score (Smith and Waterman, 1981) for local alignments. They can be computed by dynamic programming.

Tree edit distance Because of the growing interest in applications that naturally involve tree-structured data (such as the secondary structure of RNA in biology, XML documents on the web or parse trees in natural language processing), several works have extended the string edit distance to trees, resorting to the same elementary edit operations (see (Bille, 2005), for a survey on the matter). There exist two main variants of the tree edit distance that differ in the way the deletion of a node is handled. In (Zhang and Shasha., 1989), when a node is deleted all its children are connected to its father. The best algorithms for computing this distance have an $O(n^3)$ worst-case complexity, where n is the number of nodes of the largest tree (see (Pawlik and Augsten, 2011), for an empirical evaluation of several algorithms). Another variant is due to (Selkow, 1977), where insertions and deletions are restricted to the leaves of the tree. Such a distance is relevant to specific applications. For instance, deleting a $< UL >$ tag (i.e., a nonleaf node) of an unordered list in an HTML document would require the iterative deletion of the $< LI >$ items (i.e., the subtree) first, which is a sensible thing to do in this context. This version can be computed in quadratic time. Note that tree edit distance computations can be made significantly faster (especially for large trees) by exploiting lower bounds on the distance between two trees that are cheap to obtain (see for instance (Yang et al., 2005)). A study on the expressiveness of similarities and distances on trees was proposed by (Emms and Franco-Penya, 2012).

Like in the string case, there exists a few methods for learning the cost matrix of the tree edit distance (see Section 4.1.1). Note that the edit similarity learning method presented in Chapter 4, can be used for both strings and trees edit distances.

In general, trees can be divided in ordered and not ordered trees, and in evolutionary and not evolutionary trees. As the time dimension of music is represented implicitly in the left-to-right ordering of trees (see Chapter 3), we deal only with ordered trees. Regarding the evolutionary trees, they are often used to conceptually represent the evolutionary relationship of species or organisms in biology, evolution of works in linguistics, statistical classifications, or even tracking computer viruses. The metrical trees we use are not evolutionary trees because they do not have distinct labels, so those algorithms are not applicable for our problem. In Chapter 4 a similarity learning approach is described. This uses the two classical edit distances (Zhang and Shasha., 1989) and (Selkow, 1977) as initial point to learn the new similarity proposed.

Graph edit distance Note that there also exist extensions of the edit distance to general graphs (Gao et al., 2010), but like many problems on graphs, in general, computing a graph edit distance is NP-hard, making it impractical for real-world tasks.

Spectrum, subsequence and mismatch kernels These string kernels represent strings by fixed-length feature vectors and rely on explicit mapping functions ϕ. The spectrum kernel (Leslie et al., 2002) maps each string to a vector of frequencies of all contiguous subsequences of length p and computes the inner product between these vectors. The subsequence kernel (Lodhi et al., 2002) and the mismatch kernel (Leslie et al., 2003) extend the spectrum kernel to inexact subsequence matching: the former considers all (possibly noncontiguous) subsequences of length p while the latter allows a number of mismatches in the subsequences.

String edit kernels String edit kernels are derived from the string edit distance (or related measures). The classic edit kernel (Li and Jiang, 2004) has the following form:

$$K_{L\&J}(x, x') = e^{-t \cdot d_{lev}(x,x')},$$

where d_{lev} is the Levenshtein distance and $t > 0$ is a parameter. However, (Cortes et al., 2004) have shown that this function is not PSD (and thus is not a valid kernel) in the general case for nontrivial alphabets. Thus, one has to tune t, hoping to make K PSD. Moreover, it suffers from the so-called "diagonal dominance" problem (i.e., the kernel value decreases exponentially fast with the distance), and SVM is known not to perform well in this case (Schölkopf et al., 2002). A different string edit kernel was proposed by (Neuhaus and Bunke, 2006) and is defined as follows:

$$K_{N\&B}(x, x') = \frac{1}{2}(d_{lev}(x, x_0)^2 + d_{lev}(x_0, x')2 - d_{lev}(x, x')^2),$$

where x_0 is called the "zero string" and must be picked by hand. They also propose combinations of such kernels with different zero strings. However, the validity of such kernels is not guaranteed either. (Saigo et al., 2004) build a kernel from the sum of scores over all possible Smith-Waterman local alignments between two strings instead of the alignment of highest score only. They show that if the score matrix is PSD, then the kernel is valid in general. However, like $K_{L\&J}$, it suffers from the diagonal dominance problem. In practice, the authors take the logarithm of the kernel and add a sufficiently large diagonal term to ensure the validity of the kernel.

Convolution kernels The framework of convolution kernels (Haussler, 1999) can be used to derive many kernels for structured data. Roughly speaking, if structured instances can be seen as a collection of subparts, then Haussler's convolution kernel between two instances is defined as the sum of the return values of a predefined kernel over all possible pairs of subparts, and is guaranteed to be PSD. Mapping kernels (Shin and Kuboyama, 2008) are a generalization of convolution kernels as they allow the sum to be computed only over a predefined subset of the subpart pairs. These frameworks have been used to design several kernels between structured data (Collins and Duffy, 2002) (Shin and Kuboyama, 2008) (Shin et al., 2011). However, building such kernels is often not straightforward since they suppose the existence of a kernel between subparts of the structured instances.

Marginalized kernels When one has access to a probabilistic model encoding for instance the probability that a string (or a tree) is turned into another one, marginalized kernels (Tsuda et al., 2002) (Kashima et al., 2003), of which the Fisher kernel (Jaakkola and Haussler, 1998) is a special case, are a way of building a kernel from the output of such models.

3

Melody Tree Representation

3.1 Introduction

The tree structure is an alternative representation between strings and graphs. On the one hand, its expressive capacity is higher than the strings and allow describing naturally hierarchical structures in which relations between its components are given. On the other hand, its management from the theoretical point of view is much simpler and efficient than that of graphs. Trees have been used by a number of authors in the computer music literature with different aims.

Formal language theory uses trees in a natural way and Lee (Lee, 1985) tried to interpret rhythms by using grammars. Something similar did Bod (Bod, 2002), but aiming to learn how to automatically segment melodies, using the tree approach provided by parsing the melody. In the work of Conklin (Gilbert and Conklin, 2007), monodies were parsed into tree structures using a probabilistic context-free grammar to perform melodic reductions that were used also in a segmentation task.

In the context of assisted musical composition, trees have been used as a way to conceptually represent music (Balaban, 1996; Smaill et al., 1993). Under this approach, the *Wind in the Willows* system (Högberg, 2005) used tree transducers to generate music. The renowned tool for assisted composition OpenMusic (Assayag et al., 1999) uses trees as a natural way for representing the hierarchical nature of duration subdivision of musical figures and groupings like tuplets.

For representing musicological analyses, the tree representation was used in the Generative Theory of Tonal Music (GTTM) (Lerdahl and Jackendoff, 1983) and in a number of works based on the Schenkerian analysis (Kirlin and Utgoff, 2008; Marsden, 2001, 2005, 2007; Smoliar, 1979).

Finally, trees have been also used not as a means to represent music, but as an intermediate data structure for other goals like building document

structures for indexing (Blackburn, 2000; Drewes and Högberg, 2007; Skalak et al., 2008).

However, trees had not been used as a method to represent music for comparison until Rizo and Iñesta propose this approach in (Rizo and Iñesta, 2002). Despite this structure seems to be the most suitable for capturing the temporal and hierarchical structure, and it is an adequate data structure to encode and process music in symbolic format for similarity computation.

This dissertation is a continuation of the work did in (Rizo, 2010) where the metrical trees were defined. Now, we try to outfit the representation of melodies by trees with new methods to be compared and with inferred languages that can capture the relations between trees. These approaches could be used to generate new melodies or help in the composition in future works. The tree structured used in this dissertation is described in the next section. For a extended review and more extensive explanation of the tree structure in music, the reader can refer to (Rizo, 2010).

3.2 Tree Representation of Monophonic Metered Music

As it was said in Section 1.2, a melody has two main dimensions: rhythm and pitch. In linear representations, those dimensions are coded by explicit symbols, but ordered trees are able to implicitly represent time in their structure, making use of the fact that the whole piece is divided hierarchically into bars, and note durations are multiples of basic time units, mainly in a binary (sometimes ternary) subdivision. The left to right ordering of tree nodes depicts the time flow. This way, trees are less sensitive to the codes used to represent melodies, since only pitch codes are needed to be established and thus there are less degrees of freedom for coding. Furthermore, this hierarchical organization allows to append additional information such as harmonic descriptors to groups of notes in a natural way. We name this kind of representation *metrical trees* (see (Rizo, 2010)).

In the western music from the common practice period, the lapse of time is usually divided first in bars regularly, then in beats and finally in subdivisions of those beats. The duration of the bars depends on the meter and tempo. The duration of the actual figures is designed according to a logarithmic scale: a *whole* note lasts twice than a *half* note, that is two times longer than a *quarter* note, etc. (see figure durations hierarchy in Fig. 3.1 at page 33).

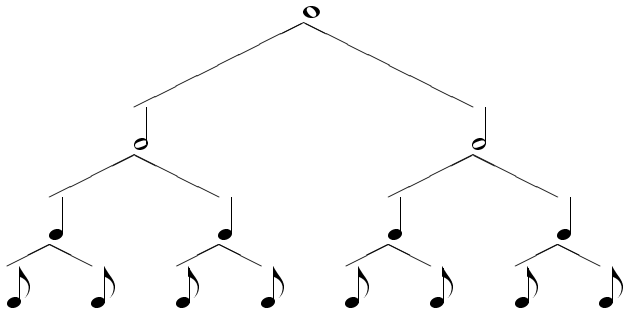


Figure 3.1: Duration hierarchy for note figures in binary meters. From top to bottom: whole (4 beats), half (2 beats), quarter (1 beat), and eighth (1/2 beat) notes.

This representation has two main drawbacks: its tight dependency on the meter structure of the input source, and its difficulty to represent tied notes whose durations are summed, dots (duration is extended in an additional 50%), and syncopations. The first problem can be overcome through an a priori metrical analysis of the work (Eck and Casagrande, 2005; Meudic, 2002) in the case that the meter metadata is not present in the data source. The second drawback, from the representation point of view, can be solved by the addition of a special symbol that encodes the concept of note continuation. In other words, when a note exceeds the considered duration, in terms of binary divisions of time, it is subdivided into notes of binary durations, and the resulting notes are coded in their proper tree levels. Nodes containing the continuation of another one can be encoded by appending any special symbol or indication to denote the continuation. For the comparison task, this is not a problem as other authors point out (Hanna et al., 2008; Mongeau and Sankoff, 1990; Pardo and Sanghi, 2005). One derived problem is the excessive growth of trees when very short notes and performance imprecisions are found in the case of real time sequenced data. This problem has been addressed by tree pruning methods and advanced quantization algorithms (Agon et al., 1994; Cemgil et al., 2000).

3.3 Melody Tree Representation Used

In this dissertation we use a tree-based symbolic representation of melodies as suggested by previous works for melody classification (Habrard et al., 2008; Rizo et al., 2009). The representation uses rhythm for defining the tree structure and pitch information for node labeling. To represent the note pitches in a monophonic melody M, we use symbols σ from a pitch

representation alphabet Σ_p. In this work, the interval from the tonic of the song modulo 12 is used as a pitch descriptor and the symbol '$-$' represents rests ($\Sigma_p = \{p \in \mathcal{N} \mid 0 \leq p \leq 11\} \cup \{-\}$). This encoding has the advantage of being octave invariant, i.e, two melodies belonging to two different octaves will be equally encoded. However, it is not able to distinguish between enharmonic tones (the notes C$\sharp$ and D$\flat$ are enharmonic). On the other hand, it can be obtained directly from the absolute pitch just by computing a modulo 12 on the original MIDI note. This way, in 'G Major', any pitch 'G' is mapped to 0. This alphabet permits a transposition invariant representation and keeps cardinality low. Rests are represented by a special label '$-$'.

In the proposed approach, each melody bar is represented by a tree, $t \in T_\Sigma$. Bars are coded by separated trees and then they are linked to a common root. The level of a node in the tree determines the duration it represents (see Fig. 3.1). The root (level 1) represents the duration of the whole bar for a binary meter, the two nodes in level 2 the duration of the two halves of a bar, etc. In general, nodes in level i represent duration of a $1/2^{i-1}$ of a bar for a binary meters ($1/3^{i-1}$ for ternary). Therefore, during the tree construction, nodes are created top-down when needed and guided by the meter, to reach the appropriate leaf level to represent a note duration. In that moment, the corresponding leaf node is labeled with the pitch representation symbol, $\sigma \in \Sigma_p$.

Once the tree has been built, a bottom-up propagation of the pitch labels is performed to label all the internal nodes (see an example in Fig. 3.2). The rules for this propagation are inspired on a melodic analysis (Illescas et al., 2007).

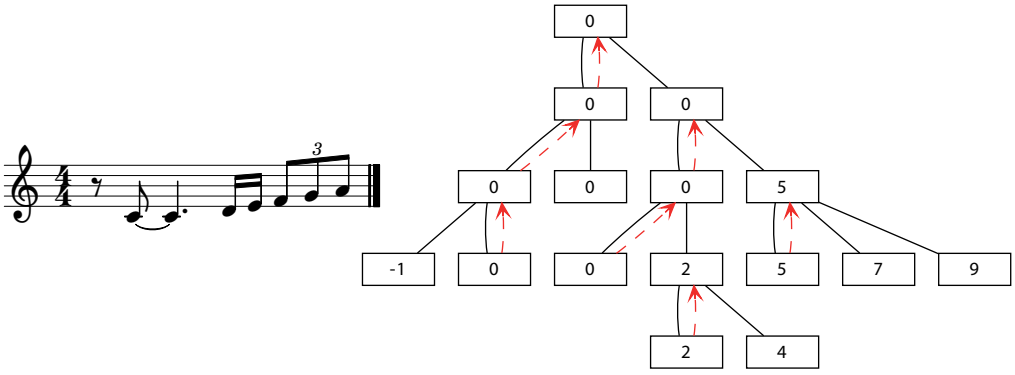


Figure 3.2: Tree representation of a one-bar melody with an example of how pitch labels are propagated.

All the notes are tagged either as *harmonic tones*, for those belonging to the current harmony at each time, or as *non-harmonic tones* for ornamental

notes. Harmonic notes have always priority for propagation and when two harmonic notes share a common father node, propagation is decided according to the metrical strength of the note (the stronger the more priority), depending on its position in the bar and the particular meter of the melody. Notes have always higher priority than rests (Fig. 3.2 shows an example). Eventually, when all the internal nodes are labeled, all bar trees are linked to a common forest root, labeled with the root of the first bar tree. In this way, the whole melody is completely represented (see Fig. 3.3 for an illustration).

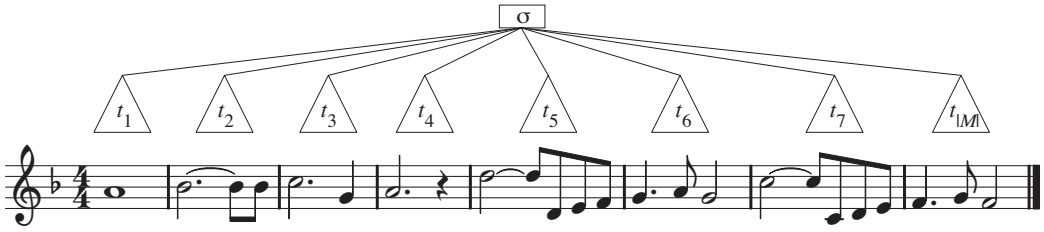


Figure 3.3: Representation of a $|M|$-bar melody.

It must be noted that our objective here is not to output an accurate tonal analysis but to use a simple, deterministic, and computable model to select the notes that seem to be more important in the melody. The complex problem of analyzing the music to be reduced has been extensively studied in works like (Gilbert and Conklin, 2007; Lerdahl and Jackendoff, 1983; Marsden, 2005).

The main motivation to label the internal nodes is that tree similarity algorithms used in this dissertation need all the nodes (both internal and leaves) to have a label, but in the metric tree construction process, after the structure of the tree is completed, the pitch labels are just in the leaves.

The set of rules are used for propagating the labels from the leaves upwards, labeling the internal nodes. The propagation of a label is decided on the basis that the note in that node is more important than that of the sibling node. The importance of a note is related to its capability to contribute to the melody identity.

The goal of this work is to measure the similarity between two pieces of music represented by trees. Thus, in order to compare music, we need a measure to compare metric trees. In the metric trees depicted so far, the tree structure describes meter and beat information. Leaves represent the actual pitches and inner nodes contain the result of the bottom-up propagation just described. The devised similarity measure should assign higher rates to the comparison of two variations of the same melody than to two different works.

These variations are supposed to be represented by similar trees in structure and pitch labels.

3.4 Corpora

In our experiments, we have tried to identify a target melody using three different corpora. One of them is a set of different variations played by musicians and the other two are different variations automatically generated. All of them used to training the models for the different approaches. In the following, we describe these three different corpora used in this dissertation.

3.4.1 Pascal Database

The first corpus, named *Pascal*[1] consisting of a set of 420 monophonic 8-12 bar incipits of 20 worldwide well known tunes of different musical genres. For each song, a canonic version was created using a score editor. Then it was synthesized and the audio files were given to three amateur and two professional musicians (a classical and a jazz player). Each musician listened to the songs (to identify the part of the tune to be played) and they played them on MIDI controllers (four keyboards, one guitar), real-time sequencing them 20 times with different embellishments and without avoiding performance errors. This way, for each of the 20 original scores, 21 different sequences were built. See appendix A.1 for the full listing of tunes included.

3.4.2 Essen Corpora

Two other corpora (*Essen-Kinder* and *Essen-Lied*) have been used for testing the proposed method in order to test its performance on other kind of variations. These corpora were constructed from the publicly available Essen Associative Code Folksong Database. The *Essen-Kinder* corpus was constructed choosing 337 melodies (MIDI encoded) from the whole Essen database[2] (Selfridge-Field, 1997) and choosing those containing the string 'Kinder' in the tag 'FCT'. From each melody, 12 variations were generated as explained below. The *Essen-Lied* corpus has 27 melodies and 64 variations,

[1] This name was given after the name of a project funded by an European Union Network of Excellence named PASCAL (Pattern Analysis, Statistical Modeling and Computational Learning).

[2] Downloaded from http://www.esac-data.org/

containing songs in the Essen database where the 'FCT' tag contains the term 'Lied'.

The different variations in this case were automatically generated using a software developed to it, that simulates different interpretation mistakes, like adjacent note substitutions, note deletions, or note insertions, driven by probabilities given as external parameters. Moreover, grid quantizations for several grooves were utilized (see an example in appendix A.2).

4

Tree Similarity Learning

In this chapter, we review the literature on supervised metric learning from structured data such as strings and trees, with an emphasis on the pros and cons of each method. Then, we study the possibilities of the tree edit similarity to classify melodies using a new approach for estimate the weights of the edit operations.

4.1 Introduction

As discussed in Section 2.3, using an appropriate metric is key to the performance of many learning algorithms. Since manually tuning metrics (when they allow some parameterization) for a given real-world problem is often difficult and tedious, a lot of work has gone into automatically learning them from labeled data, leading to the emergence of metric learning.

Generally speaking, supervised metric learning approaches rely on the reasonable intuition that a good similarity function should assign a large (resp. small) score to pairs of points of the same class (resp. different class), and conversely for a distance function. Following this idea, they aim at finding the parameters (usually a matrix) of the metric such that it best satisfies local constraints built from the training sample T. They are typically pair or triplet-based constraints of the following form:

$$S = \{(z_i, z_j) \in T \times T : x_i \text{ and } x_j \text{ should be similar}\},$$

$$D = \{(z_i, z_j) \in T \times T : x_i \text{ and } x_j \text{ should be dissimilar}\},$$

$$R = \{(z_i, z_j, z_k) \in T \times T \times T : x_i \text{ should be more similar to } x_j \text{ than to } x_k\},$$

where S and D are often referred to as the positive and negative training pairs respectively, and R as the training triplets. These constraints are usually derived from the labels of the training instances. One may consider for

instance all possible pairs/triplets or use only a subset of these, for instance based on random selection or a notion of neighborhood.

Metric learning often has a geometric interpretation: it can be seen as finding a new feature space for the data where the local constraints are better satisfied. Learned metrics are typically used to improve the performance of learning algorithms based on local neighborhoods such as k-NN.

The mainly research on metric learning have been done from feature vectors obtaining a huge number of formulations and algorithms. An extended review can be found in (Bellet et al., 2015). In this work we are interested on the metric learning mainly from structure data and specially from tree data. In the following section we show a little review of this topic.

4.1.1 Metric Learning from Structured data

As pointed out earlier, metrics have a special importance in the context of structured data: they can be used as a proxy to access data without having to manipulate these complex objects. As a consequence, given an appropriate structured metric, one can use k-NN, SVM, K-Means or any other metric-based algorithm as if the data consisted of feature vectors.

Unfortunately, for the same reasons, metric learning from structured data is challenging because most of structured metrics are combinatorial by nature, which explains why it has received less attention than metric learning from feature vectors. Most of the available literature on the matter focuses on learning metrics based on the edit distance. Clearly, for the edit distance to be meaningful, one needs costs that reflect the reality of the considered task. To take a simple example, in typographical error correction, the probability that a user hits the Q key instead of W on a QWERTY keyboard is much higher than the probability that he hits Q instead of Y. For some applications, such as protein alignment or handwritten digit recognition, well-tailored cost matrices may be available (Dayhoff et al., 1978) (Henikoff and Henikoff, 1992) (Micó and Oncina, 1998). Otherwise, there is a need for automatically learning a nonnegative $(|\Sigma| + 1) \times (|\Sigma| + 1)$ cost matrix C for the task at hand.

What makes the cost matrix difficult to optimize is the fact that the edit distance is based on an optimal script which depends on the edit costs themselves. Most general-purpose approaches get round this problem by considering a stochastic variant of the edit distance, where the cost matrix defines a probability distribution over the edit operations. One can then define an edit similarity equal to the posterior probability $p_e(x'|x)$ that an input string x is turned into an output string x'. This corresponds to summing over all possible edit scripts that turn x into x' instead of

only considering the optimal script. Such a stochastic edit process can be represented as a probabilistic model and one can estimate the parameters (i.e., the cost matrix) of the model that maximize the expected log-likelihood of positive pairs. This is done via an iterative Expectation-Maximization (EM) algorithm (Dempster et al., 1977), a procedure that alternates between two steps: an Expectation step (which essentially computes the function of the expected log-likelihood of the pairs with respect to the current parameters of the model) and a Maximization step (computing the updated edit costs that maximize this expected log-likelihood). Note that unlike the classic edit distance, the obtained edit similarity does not usually satisfy the properties of a distance (in fact, it is often not symmetric).

String Edit Metric Learning

Generative models The first method for learning a string edit metric was proposed by (Ristad and Yianilos, 1998). They use a memoryless stochastic transducer which models the joint probability of a pair $p_e(x, x')$ from which $p_e(x'|x)$ can be estimated. Parameter estimation is performed with EM and the learned edit probability is applied to the problem of learning word pronunciation in conversational speech. (Bilenko and Mooney, 2003) extended this approach to the Needleman-Wunsch Score with affine gap penalty and applied it to duplicate detection. To deal with the tendency of Maximum Likelihood estimators to overfit when the number of parameters is large (in this case, when the alphabet size is large), (Takasu, 2009) proposes a Bayesian parameter estimation of pair-HMM providing a way to smooth the estimation. Experiments are conducted on approximate text searching in a digital library of Japanese and English documents.

Discriminative models The work of (Oncina and Sebban, 2006) describes three levels of bias induced by the use of generative models: (i) dependence between edit operations, (ii) dependence between the costs and the prior distribution of strings p_e(x), and (iii) the fact that to obtain the posterior probability one must divide by the empirical estimate of $p_e(x)$. These biases are highlighted by empirical experiments conducted with the method of (Ristad and Yianilos, 1998). To address these limitations, they propose the use of a conditional transducer that directly models the posterior probability $p_e(x'|x)$ that an input string x is turned into an output string x' using edit operations. Parameter estimation is also done with EM and the paper features an application to handwritten digit recognition, where digits are represented as sequences of Freeman codes (Freeman, 1974). In order to allow the use of negative pairs, (McCallum et al., 2005) consider another

discriminative model, conditional random fields, that can deal with positive and negative pairs in specific states, still using EM for parameter estimation.

Methods based on gradient descent The use of EM has two main drawbacks: (i) it may converge to a local optimum, and (ii) parameter estimation and distance calculations must be done at each iteration, which can be very costly if the size of the alphabet and/or the length of the strings are large.

(Saigo et al., 2006) manage to avoid the need for an iterative procedure like EM in the context of detecting remote homology in protein sequences. They learn the parameters of the Smith-Waterman score which is plugged in their local alignment kernel (Saigo et al., 2004). Unlike the Smith-Waterman score, the local alignment kernel, which is based on the sum over all possible alignments, is differentiable and can be optimized by a gradient descent procedure. The objective function that they optimize is meant to favor the discrimination between positive and negative examples, but this is done by only using positive pairs of distant homologs. The approach has two additional drawbacks: (i) the objective function is nonconvex and it thus subject to local minima, and (ii) the kernels validity is not guaranteed in general and is subject to the value of a parameter that must be tuned. Therefore, the authors use this learned function as a similarity measure and not as a kernel.

Tree Edit Metric Learning

Bernard et al. Extending the work of (Ristad and Yianilos, 1998) and (Oncina and Sebban, 2006) on string edit similarity learning, (Bernard et al., 2006) (Bernard et al., 2008) propose both a generative and a discriminative model for learning tree edit costs. They rely on the tree edit distance by (Selkow, 1977) - which is cheaper to compute than that of (Zhang and Shasha., 1989) - and adapt the updates of EM to this case. An application to handwritten digit recognition is proposed, where digits are represented by trees of Freeman codes.

Boyer et al. The work of (Boyer et al., 2007) tackles the more complex variant of the tree edit distance (Zhang and Shasha., 1989), which allows the insertion and deletion of single nodes instead of entire subtrees only. Parameter estimation in the generative model is also based on EM, and the usefulness of the approach is illustrated on an image recognition task.

Neuhaus and Bunke In their paper, (Neuhaus and Bunke, 2007) learn a (more general) graph edit similarity, where each edit operation is modeled by a Gaussian mixture density. Parameter estimation is done using an EM-like algorithm. Unfortunately, the approach is intractable: the complexity of the EM procedure is exponential in the number of nodes (and so is the computation of the distance).

Dalvi et al. The work of (Dalvi et al., 2009) points out a limitation of the approach of (Bernard et al., 2006) (Bernard et al., 2008): they model a distribution over tree edit scripts rather than over the trees themselves, and unlike the case of strings, there is no bijection between the edit scripts and the trees. Recovering the correct conditional probability with respect to trees requires a careful and costly procedure. They propose a more complex conditional transducer that models the conditional probability over trees and use EM for parameter estimation. They apply their method to the problem of creating robust wrappers for webpages.

Emms The work of (Emms, 2012) points out a theoretical limitation of the approach of (Boyer et al., 2007): the authors use a factorization that turns out to be incorrect in some cases. Emms shows that a correct factorization exists when only considering the edit script of highest probability instead of all possible scripts, and derives the corresponding EM updates. An obvious drawback is that the output of the model is not the probability $p_e(x'|x)$. Moreover, experiments on a question answering task highlight that the approach is prone to overfitting, and requires smoothing and other heuristics (such as a final step of zeroing-out the diagonal of the cost matrix).

This review raises two observations:

1. There is a relatively small body of work on metric learning from structured data, presumably due to the higher complexity of the learning procedures. Almost all existing methods are based on probabilistic models: they are trained using an expensive iterative algorithm and cannot accommodate negative pairs. Furthermore, no approach is guaranteed to converge to the global optimum of the optimized quantity and again, there is a lack of theoretical study.

2. The use of learned metrics is typically restricted to algorithms based on local neighborhoods, in particular k-NN classifiers. Since the learned metrics are typically optimized over local constraints, it seems unclear whether they can be successfully used in more global classifiers such as SVM and other linear separators, or if new metric learning algorithms should be designed for this global setting. Furthermore, building a PSD kernel from the learned

metrics is often difficult, especially for structured data (e.g., string edit kernels).

4.2 Background

We begin this part by making a little recap on the definition of a tree.

Definition 10 *Let $t = \sigma(t_1, \ldots, t_n)$ be a tree where σ is a node and $t_1, \ldots, t_n$ is an ordered sequence of (sub)trees. σ is called the root of t. $t_1, \ldots, t_n$ are called the children of σ. A node with no child ($n = 0$) is a leaf.*

The size of the tree t is denoted by $|t|$ and corresponds to the number of nodes constituting the tree. In the following, we assume each node to be labeled by a symbol σ coming from a set of labels Σ. We denote by T_Σ the set of trees that can be built from Σ.

In this work, we stand in a classical supervised learning setting for binary classification. We assume we are given some labeled examples $z = (t, \ell)$ drawn from an unknown distribution P over $T_\Sigma \times \{-1, 1\}$, where $-1, 1$ are the binary labels of the examples. $K : T_\Sigma \times T_\Sigma \to [-1, 1]$ defines a pairwise similarity function over T_Σ, K is symmetric if for all $t, t' \in T_\Sigma$, $K(t, t') = K(t', t)$. A binary classifier h is a function $h : T_\Sigma \to \{-1, 1\}$.

The binary classifiers we consider rely on a similarity function that is based on the notion of edit distance over trees. The tree edit distance (TED) (Bille, 2005) can be seen as an extension or a generalization of the string edit distance (also known as the Levenshtein distance) and is based on three elementary edit operations: substitution, insertion or deletion of nodes. It is defined as follows.

Definition 11 *Let Σ be a set of labels and \$ be the empty symbol, T_Σ being the set of all possible trees (including the empty tree \$). The tree edit distance (TED) $e_T(t, t')$ between two trees t and t' is the minimum number of edit operations to change t into t'. The set corresponding to the minimum number of operations allowing one to change t into t' is called the optimal script.*

Like in the case of strings, TED can be computed using dynamic programming: When considering two trees of sizes m and n, where $m \leq n$, the best known algorithms for this problem, due to (Zhang and Shasha., 1989) and (Klein, 1998), have an $O(n^3 \log n)$ time complexity and an $O(mn)$ space complexity. In these approaches, when a tree node is deleted, all its children are connected to its father. A less costly variant of these algorithms has been proposed by (Selkow, 1977), where deleting a node leads to the removal of the

entire (sub)tree rooted at that node (See Figure 4.1 for an illustration). The insertion of a (sub)tree is also allowed and requires the iterative insertion of its nodes. Such a distance is relevant to specific applications. For instance, it would make no sense to delete a <UL> tag (i.e., a node) of an unordered list in an HTML document without removing the <LI> items (i.e., the subtree). In this case, the tree edit distance can be computed by dynamic programming in quadratic time. See (Bille, 2005) for a more extended explanation of tree edit distances.

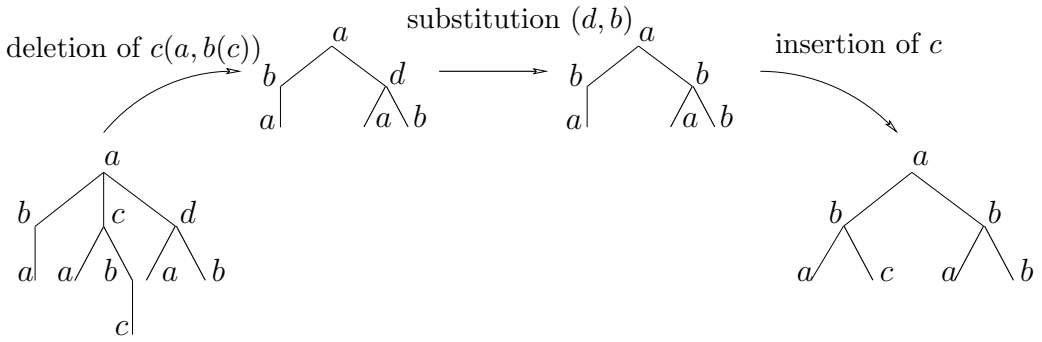


Figure 4.1: An optimal edit script according to Selkow tree edit distance algorithm (Selkow, 1977). The script begins by the deletion of the subtree $c(a, b(c))$, then it considers the substitution of two labeled nodes (d, b) and terminates with the insertion of the one node subtree c. Following this script between the two trees, the non zero entries of the corresponding matrix # of Section 4.3.1 are: $\#_{(a,\$)}$, $\#_{(b,\$)}$, $\#_{(c,\$)}$, $\#_{(\$,c)}$, and $\#_{(d,b)}$; all these entries receive the value 1 since they are exactly used once in the script.

In the next section, we present the theory of (ϵ, γ, τ)-good similarity functions, allowing one to learn low error binary classifiers using good similarity functions.

4.2.1 Framework for Learning with *Good* Similarity Functions

Recently, Balcan et al (2006; 2008) introduced a new theory for learning with *good* similarity functions. Their motivation was to overcome two major limitations of kernel theory. Balcan theory relaxes two major drawbacks of kernel theory used for learning SVM-based classifiers. First, a similarity K must be symmetric and positive semi-definite to define a kernel, these requirements often rules out natural similarity functions for the problem at

hand. Second, a notion of good kernel is not intuitive in that it is defined according to an implicit possibly unknown projection space making hard the design of good kernels for a given problem forgetting that a good kernel is essentially a good similarity function. To overcome these drawbacks, Balcan et al. (2008) proposed the following definition of *good similarity function*.

Definition 12 (Balcan et al., 2008) *A similarity function $K : X \times X \to [-1, 1]$ is an (ϵ, γ, τ)-good similarity function for a learning problem P if there exists a (random) indicator function $R(x)$ defining a (probabilistic) set of "reasonable points" such that the following conditions hold:*

1. *A $1 - \epsilon$ probability mass of examples (x, ℓ) satisfy*

$$\mathbf{E}_{(x',\ell')\sim P}[\ell\ell' K(x, x')|R(x')] \geq \gamma. \tag{4.1}$$

2. $\Pr_{x'}[R(x')] \geq \tau$.

The first condition is essentially requiring that *a $1 - \epsilon$ proportion of examples x are on average more similar to reasonable examples of the same class than to reasonable examples of the opposite class by a margin γ* and the second condition that *at least a τ proportion of the examples are reasonable*.

Yet Definition 12 is very interesting in three respects. First, it is a strict generalization of the notion of good kernel (Balcan et al., 2008) but does not impose positive semi-definiteness nor symmetry. Second, as opposed to pair and triplet-based criteria used in metric learning, Definition 12 is based on an average over some points. In other words, it relaxes the notion of local constraints, opening the door to metric learning for global algorithms. Third, these conditions are sufficient to learn well, i.e., to induce a classifier with low true risk, as we show in the following.

Let K be an (ϵ, γ, τ)-good similarity function. If the set of reasonable points $R = (x'_1, y'_1), (x'_2, y'_2), \ldots, (x'_{|R|}, y'_{|R|})$ is known, it follows directly from Equation 4.1 that the following classifier achieves true risk at most ϵ at margin γ:

$$h(x) = sign\left[\frac{1}{|R|}\sum_{i=1}^{|R|} y'_i K(x, x'_i)\right].$$

Note that h is a linear classifier in the space of the similarity scores to the reasonable points. In other words, K is used to project the data into a new space using the mapping $\phi : \chi \to R^{|R|}$ defined as:

$$\phi_i(x) = K(x, x'_i),\ i \in \{1, \ldots, |R|\}.$$

However, in practice the set of reasonable points is unknown. We can get around this problem by sampling points (called landmarks) and use them

to project the data into a new space (using the same strategy as before).If we sample enough landmarks (this depends in particular on τ, which defines how likely it is to draw a reasonable point), then with high probability there exists a linear classifier in that space that achieves true risk close to ϵ. This is formalized in Theorem 4.

Theorem 4 *(Balcan et al., 2008). Let K be an (ϵ, γ, τ)-good similarity function for a learning problem P. Let $L = x'_1, x'_2, \ldots, x'_{n_L}$ be a sample of $n_L = \frac{2}{\tau}\left(\log(2/\delta) + 8\frac{\log(2/\delta)}{\gamma^2}\right)$ landmarks drawn from P. Consider the mapping $\phi^L : X \to R^{n_L}$ defined as follows: $\phi^L_i(x) = K(x, x'_i)$, $i \in \{1, \ldots, n_L\}$. Then, with probability at least $1 - \delta$ over the random sample L, the induced distribution $\phi^L(P)$ in R^{n_L} has a linear separator of error at most $\epsilon + \delta$ relative to L_1 margin at least $\gamma/2$.*

Unfortunately, finding this separator is NP-hard (even to approximate) because minimizing the number of L_1 margin violations is NP-hard. To overcome this limitation, the authors considered the hinge loss as a surrogate for the 0/1 loss (which counts the number of margin violations) in the following reformulation of Definition 12.

Definition 13 *(Balcan et al., 2008). A similarity function K is an (ϵ, γ, τ)-good similarity function in hinge loss for a learning problem P if there exists a (random) indicator function $R(x)$ defining a (probabilistic) set of reasonable points such that the following conditions hold:*

1. $E_{(x,y)\sim P}[[1 - yg(x)/\gamma]_+] \leqslant \epsilon$, *where* $g(x) = E_{(x',y')\sim P}[y'K(x, x')|R(x')]$,
2. $Pr_{x'}[R(x')] \geqslant \tau$.

This leads to the following theorem, similar to Theorem 4.

Theorem 5 *(Balcan et al., 2008). Let K be an (ϵ, γ, τ)-good similarity function in hinge loss for a learning problem P. For any $\epsilon_1 > 0$ and $0 \leqslant \delta \leqslant \gamma\epsilon_1/4$, let $L = \{x'_1, x'_2, \ldots, x'_{n_L}\}$ be a sample of $n_L = \frac{2}{\tau}\left(\log(2/\delta) + 16\frac{\log(2/\delta)}{\epsilon_1\gamma^2}\right)$ landmarks drawn from P. Consider the mapping $\phi^L : X \to R^{n_L}$ defined as follows: $\phi^L_i(x) = K(x, x'_i)$, $i \in \{1, \ldots, n_L\}$. Then, with probability at least $1 - \delta$ over the random sample L, the induced distribution $\phi^L(P)$ in R^{n_L} has a linear separator of error at most $\epsilon + \epsilon_1$ at margin γ.*

The objective is now to find a linear separator $\alpha \in R^{n_L}$ that has low true risk based on the expected hinge loss relative to L_1 margin γ:

$$\mathbf{E}_{(x,\ell)\sim P}[[1-\ell\langle\boldsymbol{\alpha},\phi^L(x)\rangle/\gamma]_+],$$

Using a landmark sample $L = \{x'_1, x'_2, \ldots, x'_{n_L}\}$ and a training sample $T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, one can find this separator α efficiently by solving the following linear program (LP):

$$\min_{\boldsymbol{\alpha}} \sum_{i=1}^{d_l} \left[1 - \sum_{j=1}^{d} \alpha_j \ell_i K(x_i, x'_j)\right]_+ + \lambda\|\boldsymbol{\alpha}\|_1. \tag{4.2}$$

In practice, we simply use the training examples as landmarks. In this case, learning rule 4.2 referred to as Balcan's learning rule in the rest of this document - is reminiscent of the standard SVM formulation, with three important differences. First, recall that K is not required to be PSD nor symmetric. Second, the linear classifier lies in an explicit projection space built from K (called an empirical similarity map) rather than in a possibly implicit Hilbert Space induced by a kernel. Third, it uses L_1 regularization, inducing sparsity in α and thus reducing the number of landmarks the classifier is based on, which speeds up prediction. This regularization can be interpreted as a way to select (or approximate) the set of reasonable points among the landmarks: in a sense, R is automatically worked out while learning α. Note that we can control the degree of sparsity of the linear classifier: the larger λ, the sparser α. To sum up, the performance of the linear classifier theoretically depends on how well the similarity function satisfies Definition 12.

4.3 Good Edit Similarity Learning for tree-structured data

We now present our procedure for learning good tree edit similarity. This procedure follows the principle of the algorithm GESL introduced by (Bellet et al., 2012) which is based on three main steps. In this section, we detail each step of this process. First, we define a simple edit similarity based on edit scripts. Second, we optimize this similarity according to the framework of (ϵ, γ, τ)-good similarity of (Balcan et al., 2008). This step leads to good similarities allowing one to build, during a last step, powerful linear separators.

4.3.1 Tree Edit Script Based Similarity

Let C be a positive cost matrix of size $(|\Sigma|+1)\times(|\Sigma|+1)$ defining the possible edit operations over nodes of trees. $C_{i,j}$ gives the cost of the operation changing the symbol c_i into c_j, c_i and $c_j \in \Sigma \cup \{\$\}$. Let $\#(t,t')$ be an $(|\Sigma|+1)\times(|\Sigma|+1)$ matrix whose elements $\#_{i,j}(t,t')$ correspond to the number of times each edit operation consisting in changing c_i into c_j is used to turn t into t' in the optimal script obtained from any tree edit distance variant (Selkow, 1977; Zhang and Shasha., 1989). In other words, we consider the script corresponding to the minimum set of tree edit operations allowing one to change one tree into another one with respect to the tree edit distance considered (See Figure 4.1 for an illustration).

From these two matrices, the following edit function is then considered:

$$e_C(t,t') = \sum_{0\leq i,j\leq \Sigma} C_{i,j} * \#_{i,j}(t,t'). \tag{4.3}$$

An important point here is that the computation of e_C does not depend on the optimal script with respect to C. In other words, e_C is evaluated by considering the operations used in the optimal script, weighted by the custom costs C. Therefore, since the edit script defined by $\#(t,t')$ is fixed, $e_C(t,t')$ is nothing more than a linear function of the edit costs and can be optimized directly. (Bellet et al., 2012) define then the edit similarity function as:

$$K_C(t,t') = 2e^{-e_C(t,t')} - 1. \tag{4.4}$$

The use of the exponential allows K_C to introduce non-linear information in the model and to belong to $[-1,1]$ as required by Balcan et al.'s framework presented in Section 4.2.1.

4.3.2 Learning *Good* Similarity Functions

In this section, we present our algorithm for learning good edit similarities. We first define the notion of good similarity. It relies on a relaxed version of Balcan et al.'s framework (Balcan and Blum, 2006; Balcan et al., 2008).

Definition 14 (Relaxed good tree edit similarity) *A similarity function $K : T_\Sigma \times T_\Sigma \to [-1,1]$ is an (ϵ,γ,τ)-good similarity function for a learning problem P if there exists a (random) indicator function $R(t)$ defining a (probabilistic) set of "reasonable trees" such that the following conditions hold:*

1. *A* $1-\epsilon$ *probability mass of examples* (t, ℓ) *satisfy*

$$\mathbf{E}_{(t,\ell)}\left[\mathbf{E}_{(t',\ell')}\left[[1-\ell\ell' K_C(t,t')/\gamma]_+ \,|R(t')\right]\right] \leq \epsilon'. \tag{4.5}$$

where $[1-c]_+ = \max(0, 1-c)$ *corresponds to the hinge loss.*

2. $\Pr_{t'}[R(t')] \geq \tau$.

The first condition is essentially requiring that *on average a* $1-\epsilon$ *proportion of trees* t *are* 2γ *more similar to random reasonable trees of the same class than to random reasonable trees of the opposite class*[1] and the second condition that *at least a* τ *proportion of the trees are reasonable*. Note that the reasonable instances are either provided by the application at hand (in this case, they can be viewed as prototypes) or are automatically selected from a set of so-called landmarks by solving a simple linear problem which is essentially a 1-norm SVM problem (see (Balcan et al., 2008) for details - in this case, they can be viewed as support vectors).

Definition 14 is very interesting in two respects. First, it does not impose strong constraints on the form of the similarity functions considered. Second, these conditions are sufficient to learn a good linear space, i.e., to induce a linear separator $\boldsymbol{\alpha}$ in the space of the similarities to the reasonable trees: $h(\cdot) = \sum_{(t',\ell')\sim P|R(t')} \ell' K(\cdot, t')$, as illustrated in Figure 4.2 (see (Balcan et al., 2008) for a formal proof).

[1]The original definition of Balcan et al. (2006; 2008) requires the property 1 to be true only on average over the reasonable examples. However, the use of the exponential in our similarity implies a non convex formulation of the optimization problem considered for learning the costs. As a consequence, we propose to relax the original formulation by considering the surrogate version presented in Definition 14.

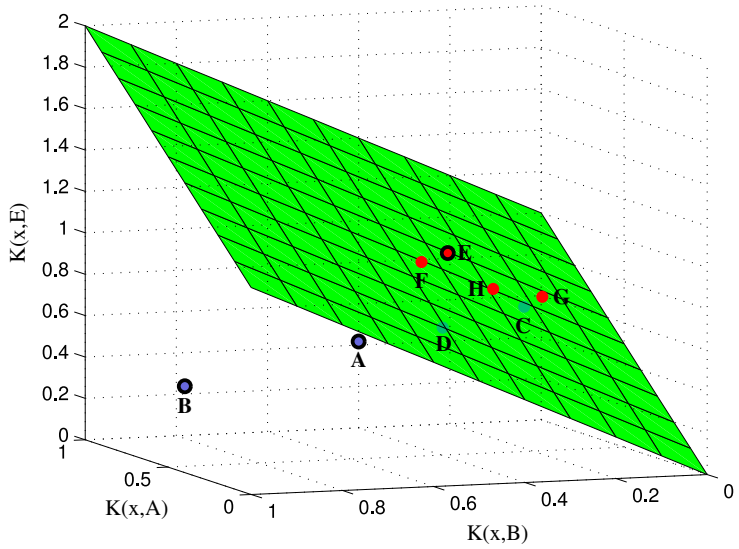


Figure 4.2: Linear separator $\boldsymbol{\alpha}$ learned from a training set of trees $\{(A,+1),(B,+1),(C,+1),(D,+1),(E,-1),(F,-1),(G,-1),(H,-1)\}$ thanks to an (ϵ,γ,τ)-good similarity function. A, B and E are reasonable trees. The good linear separator is learned in the 3D-space of the similarities to that reasonable points with respect to K.

The algorithm GESL has for objective to optimize the goodness of the similarity K_C so as to satisfy Condition 4.5 over all the pairs (z_i, z_j) of a training set of N_T labeled examples $T = \{z_i = (t_i, \ell_i)\}_{i=1}^{N_T}$. The optimization problem GESL for learning the costs C, which takes the form of a sparse convex program, is expressed as follows:

$$
\begin{array}{lll}
(GESL) & \min\limits_{C,B_1,B_2} & \frac{1}{N_T^2} \sum\limits_{1\leq i,j\leq N_T} V(C,z_i,z_j) \quad + \quad \beta\|C\|_{\mathcal{F}}^2 \\
& s.t. & V(C,z_i,z_j) = \left\{ \begin{array}{l} [B1 - e_C(t_i,t_j)]_+ \text{ if } \ell_i \neq \ell_j \\ [e_C(t_i,t_j) - B2]_+ \text{ if } \ell_i = \ell_j \end{array} \right. \\
& & B_1 \geq -\log(\tfrac{1}{2}), \quad 0 \leq B_2 \leq -\log(\tfrac{1}{2}), \quad B_1 - B_2 = \eta_\gamma \\
& & C_{i,j} \geq 0, \quad 0 \leq i,j \leq A,
\end{array}
$$

where $\beta \geq 0$ is a regularization parameter on edit costs, $\|\cdot\|_{\mathcal{F}}$ denotes the Frobenius norm and $\eta_\gamma \geq 0$ a parameter corresponding to the desired "margin". The relationship between the margin γ and η_γ is given by $\gamma = \frac{e^{\eta_\gamma}-1}{e^{\eta_\gamma}+1}$. The loss V essentially penalizes the violations of the goodness defined by Equation 4.5.

Then, we optimize the costs of these edit operations by solving this optimization problem. This allows us, once again, to avoid using a costly iterative procedure: We only have to compute the tree edit script between two trees once, which dramatically reduces the algorithmic complexity of the learning algorithm. Moreover, the procedure of GESL allows one to learn good tree edit similarities with respect to Definition 14 implying the ability to learn good linear separators for tree structured data which is presented in the next section.

4.3.3 Classifier Learning: Automatic Selection of the Reasonable Trees

As mentioned previously, reasonable points play an important role because the optimal linear classifier, named $\boldsymbol{\alpha}$, is built in the space of the similarities to that points. In areas where some ground truth is available, these reasonable points can be given by an expert. In this case, provided that we have access to a similarity that satisfies the goodness property of Definition 14, the optimal classifier is directly obtained by using the similarities to that points. In complex domains such as in music recognition, setting the set of reasonable points is a tricky task. To overcome this problem, a strategy, as suggested by Balcan et al. (2008), consists in using d (unlabeled) examples as landmarks, d_l labeled examples and in solving the following simple linear problem which is essentially a 1-norm SVM problem:

$$\min_{\boldsymbol{\alpha}} \sum_{i=1}^{d_l} \left[1 - \sum_{j=1}^{d} \alpha_j \ell_i K(t_i, t'_j) \right]_+ + \lambda \|\boldsymbol{\alpha}\|_1, \tag{4.6}$$

where $\|\cdot\|_1$ corresponds to the L_1-norm.
An important feature of (4.6) is the L_1-regularization on $\boldsymbol{\alpha}$, which induces sparsity. Therefore, it allows automatic selection of reasonable points controlling the sparsity of the solution: the larger λ, the sparser $\boldsymbol{\alpha}$. Moreover, by solving (4.6), we directly get a good (with generalization guarantees) linear separator in the space of the similarities to the reasonable points. In the experimental section, we will show that these points provide valuable information.

We present in the next section the formalism used for tree representation of melodies and we use the properties of GESL to derive theoretical guarantees in this context.

4.4 Tree-structured Representation of Melodies and Theoretical Guarantees

We use a tree-based symbolic representation of melodies as suggested by previous works for melody classification (Habrard et al., 2008; Rizo et al., 2009). The representation uses rhythm for defining the tree structure and pitch information for node labeling. Bars are coded by separated trees and then they are linked to a common root (see Figs. 3.2 and 3.3 for an illustration). The level of a node in the tree determines the duration it represents (see Section 3.3 for details).

This tree representation of melodies defines a particular class of data for which we can derive a consistency theorem tailored to our method. For any bar of duration d_{bar}, the duration of a note encoded by a node v in a tree is defined according to its depth in the tree: $\text{nodedur}(v) = \frac{1}{2^{depth(v)}} \times d_{bar}$, where $\text{depth}(v)$ denotes the depth of the node v. In order to limit the depth of the tree, a minimum note duration is generally fixed (Rizo, 2010) and defined by a constant Min_d. Symmetrically, the depth of a node v encoding a note of duration d is given by: $\text{depth}(v) = \log_2(\frac{d_{bar}}{d})$. If Max_d is the maximum duration of a bar, we have for any bar of duration d_{bar} and any note of duration d encoded by a node v,

$$\text{depth}(v) = \log_2(\frac{d_{bar}}{d}) \leq \log_2\left(\frac{\text{Max}_d}{\text{Min}_d}\right). \tag{4.7}$$

The depth of the tree represents the number of nodes on the path from the root to the node and needed to encode the note in the tree; $\log_2\left(\frac{\text{Max}_d}{\text{Min}_d}\right)$ represents the maximum node depth and thus the maximum number of nodes needed to encode a note in a tree. Now, if N_{maxb} is the maximum number of bars in a melody, $\frac{\text{Max}_d}{\text{Min}_d}$ gives the maximum number of notes in a bar and then we can bound the maximum size of a tree encoding a melody by:

$$N_{maxb} \times \frac{\text{Max}_d}{\text{Min}_d} \times \log_2\left(\frac{\text{Max}_d}{\text{Min}_d}\right). \tag{4.8}$$

Then, we have the following generalization bound for the GESL tree edit version. Indeed, one can relate the true loss of the learned matrix C_T, $L(C_T) = \mathbb{E}_{(z,z')}[V(C_T, z, z')]$, with its empirical expected loss $L_T(C_T)$ (over the pairs of the set T) with the following theorem.

Theorem 6 *Let N_{maxb}, Max_d and Min_d be respectively the maximum number of bars, the maximum duration of bars and the minimum duration*

of a note in the melodies. Let T be a sample of N_T trees drawn i.i.d. from an unknown distribution P, C_T the matrix learned by GESL (using all the possible pairs from T), with probability $1-\delta$ we have the following bound for $L(C_T)$:

$$L(C_T) \leq L_T(C_T) + 2\frac{\kappa}{N_T} + \left(2\kappa + (3B_\gamma\left(\frac{2N_{notes}}{\sqrt{\beta B_\gamma}} + 3\right)\right)\sqrt{\frac{\ln(2/\delta)}{2N_T}}, \quad (4.9)$$

with $N_{notes} = N_{maxb} \times \frac{Max_d}{Min_d} \times \log_2\left(\frac{Max_d}{Min_d}\right)$, $\kappa = \frac{6N_{notes}^2}{\beta}$ *and* $B_\gamma = \max(\eta_\gamma, -\log(1/2))$.

The proof follows the same principle as in (Bellet et al., 2012), and uses the fact that the maximum number of nodes in a tree is bounded by N_{notes}.

The previous result highlights important aspects of our method. First, it has a convergence rate in $O(\sqrt{1/N_T})$ which a classical rate for concentration bounds. Another point is that the method is independent from the alphabet size - *i.e.* from the pitch representation - which indicates that the method should scale well with large alphabets. Lastly, by considering the framework of Section 4.3.2, it provides a consistency justification of our approach by showing that the goodness of the similarity is ensured, which implies the existence of a good linear classifier in the space defined by the reasonable trees.

One drawback of this result is the dependency on a given maximum tree size N_{notes}. Fortunately, this constraint can be relaxed by assuming the trees to be generated from probabilistic models where the probability to generate trees with size larger than an integer k is bounded, like probabilistic finite tree automata. Indeed, it can be shown that with probability $1-\delta$ for any sample of N_T trees, we have for any tree t belonging to this sample,

$$|t| < \frac{\log(N_T U/\delta)}{\log(1/\rho)}, \quad (4.10)$$

with $U > 0$ and $0 < \rho < 1$ some constants (Denis et al., 2008). By combining this result with the previous theorem, one can obtain a bound independent from the maximum tree size, see (Bellet et al., 2012) for the technical details.

4.5 Experiments in Melody Recognition

In this section, we provide an experimental evaluation of GESL for tree edit distance learning. Our objective is two-fold: First, we show that this

approach allows us to learn good similarities leading to very accurate linear classifiers. Second, we illustrate how the notion of reasonable points can be used in this framework to extract semantic information from the learned similarity.

4.5.1 Pascal database

To perform our experiments, we focus on a standard task in music information retrieval, namely *melody classification*. We evaluate GESL using the *Pascal* databaseconsisting of a set of 420 monophonic 8-12 bar incipits of 20 worldwide well known tunes of different musical genres (see 3.4.1 for details).

4.5.2 Experimental setup

A three-fold cross-validation scheme is carried out to perform the experiments, where 2/3 of the database is used for training and 1/3 for testing. We compare five edit similarities: (i) the *Selkow* tree edit distance (Selkow, 1977), which constitutes the baseline, (ii) a stochastic version of the Selkow distance, Selkow$_{sto}$ (Bernard et al., 2008) learned from an EM-like algorithm based on the software SEDiL (Boyer et al., 2008), (iii) the Zhang-Shasha tree edit distance (Zhang and Shasha., 1989), (iv) K_{Selkow}, learned by GESL using the Selkow edit scripts, and (v) $K_{Zhang-Shasha}$, learned by GESL using the Zhang-Shasha edit scripts. Note that it was not possible to evaluate the stochastic version of the Zhang-Shasha distance learned with SEDiL because the learning procedure was intractable, that confirms our claim mentioned in the introduction.

Let us remind that GESL is specifically dedicated to optimize similarities that are then efficiently used in the learning of a linear separator (by solving problem (4.6) of Section 4.3.3). Therefore, in a first series of experiments, the melodies are classified using this linear separator, learned from the five similarity measures (Selkow, Selkow$_{sto}$, Zhang-Shasha, K_{Selkow} and $K_{Zhang-Shasha}$)[2]. Even though GESL has not been designed for nearest-neighbor-like algorithms, we perform a second series of experiments where the melodies are classified with the 1-nearest neighbor rule by directly making use of the five similarities.

To deal with the multi-class setting, we use a standard one versus one procedure: A binary linear classifier is learned for each pair of classes (C_j, C_k), that is, the binary linear classifier $h_{(j,k)}$ is learned considering the instances

[2]Note that the distances are used as a similarities and normalized to stand in the interval $[-1, 1]$.

Table 4.1: Success rates (%) and standard deviation obtained from the five edit similarities in 1NN and linear classifications on the *Pascal* corpus.

Approach	Success rate Linear Classifier	Success rate 1-NN
Selkow	90.2 ± 1.2	90.5 ± 0.9
Selkow$_{sto}$	88.6 ± 0.9	91.5 ± 0.4
Zhang-Shasha	91.9 ± 0.9	93.6 ± 0.5
K_{Selkow}	93.1 ± 0.5	90.5 ± 0.2
$K_{Zhang-Shasha}$	95.0 ± 0.7	90.2 ± 1.4

of the class C_j (resp. C_k) as positive (resp. negative) data. Therefore, we learn $\binom{n}{2}$ (binomial coefficient) binary classifiers, where n is the number of classes (in our problem 20 classes and 190 binary classifiers). Each classifier determines if a melody belongs to the class C_j or to the class C_k. Then, we apply a majority vote strategy to decide the final classification for each melody.

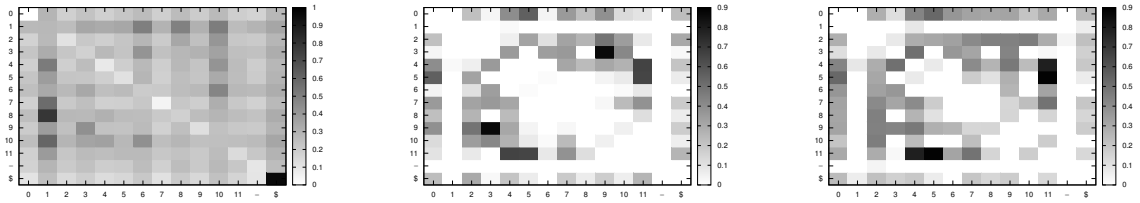
4.5.3 Results and edit cost analysis

Results are reported in Table 4.1. We can make the following remarks.

- Using a linear classifier, the similarities learned by GESL allow significant improvements of the classification accuracy. $K_{Zhang-shasha}$ achieves the best overall performance (95.0%) improving the results mentioned in (Habrard et al., 2008) for the same dataset. These results confirm the interest to optimize with GESL a tree edit distance (according to the goodness criterion of Definition 14) which is then used to learn a simple linear separator in the space of the similarities to the reasonable points.

- In the second series of experiments, we can note that the tree edit distance $Zhang - Shasha$ allows us to reach the best result (however, smaller than 95.0%). Even though K_{Selkow} and $K_{Zhang-Shasha}$ have not been optimized for a nearest neighbor classifier, it is worth noting that the tree edit cost learning procedure of GESL remains competitive with the two others.

In order to analyze the understandability of the inferred models, let us compare the edit cost matrices learned on the one hand with GESL for both Selkow and Zhang & Shahsha tree edit distances, and on the other hand



(a) SEDiL (Selkow). (b) GESL (Selkow). (c) GESL (Zhang and Shasha).

Figure 4.3: Learned edit costs by GESL and SEDiL algorithms.

with SEDiL (only for Selkow, because Zhang and Shasha's is intractable). In order to represent graphically these matrices, they have been averaged and normalized over the three folds. Our aim here is to see if the learned edit costs are able to model the changes and mistakes that can be found in the corpus. Mistakes correspond in general to note elisions, grace notes insertions, the substitution of the intended note by another at a distance of a semitone or a tone, and note duration and onset changes.

The matrix learned by SEDiL is shown in Fig. 4.3(a), while matrices learned by GESL are presented in Fig. 4.3(b) and (c). We can see that in the edit cost matrix learned with SEDiL from an EM-like algorithm, small costs (represented by bright cells) are mainly located in the diagonal of the matrix. This means that substitutions of a pitch by itself are favored. However, this reduces the ability of the model to adapt to small variations caused by the musicians. In this case, the model will prefer to delete and insert a pitch (the last row and last column contain indeed brighter cells too) to fit the canonical version. We claim that this does not model well the reality. Unlike matrix (a), the edit cost matrices (b) and (c) learned with GESL allow some distortions. We can see that the smallest costs are found not only in the substitution of a pitch by itself (the diagonal) but also in substitution of a pitch by a close pitch of about one or two semitones. This corresponds to the substitution of the intended note by another at a distance of a semitone or a tone caused by mistakes of the musicians when they played the songs.

Moreover, in both plots issued from GESL, we can identify two bright row and column that correspond to the operation of replacing a symbol σ by the rest symbol '$-$' ($(\sigma \rightarrow -)$ or $(- \rightarrow \sigma)$). It is worth noting that this perfectly models the insertions or deletions of rests by the musicians when they want to produce some effect to the melody feeling.

4.5.4 Reasonable points analysis

Finally, we provide a brief analysis of the reasonable points automatically selected by solving problem (4.6) of Section 4.3.3. Intuitively, these representative points should be some discriminative prototypes the classifier is based on. To make the analysis easier, we consider a restricted task where the goal is to predict if a melody belongs to the classical music genre or if it is a children's song. These two styles correspond to two classes allowing us to turn the task into a binary classification problem. The examples of the classical genre correspond to songs belonging to the classes 'Toccata and fugue', 'Avemaria', 'Ode to joy', 'Bolero' and 'Lohengrin, wedding march' of the *Pascal* corpus. The children's class is formed by the songs coming from 'Alouette', 'Oh! Susanna', 'Happy birthday', 'Twinkle twinkle little star' and 'Jingle bells' classes. From these data, we build a learning and a test sample such that there are 7 instances for each *Pascal* class in the test and 14 in the training set. Therefore, each class has 35 examples in test and 70 for training. Table 4.2 shows the number of reasonable points for each class. We can see that (beyond a high accuracy) in the classical class there are 4 reasonable points that belong to 'Toccata and fugue' and 5 to 'Lohengrin, wedding march'. These two songs seem thus to be good representatives for that class. In the same way, the 'Oh! Susanna' and 'Happy birthday' songs provide many (9 out of 15) discriminative prototypes for the children class. All in all, these four songs provide about 62% (18 out of 29) of the reasonable points while they represent only 40% of the training songs. Said differently, the predicted label of a new song will be mainly defined by its similarity to those 4 songs.

4.6 Conclusions

In this work, we investigated a new framework for learning tree edit distances thanks to a convex optimization problem based on the framework of GESL, originally developed for strings. We have shown that this framework is adequately tailored to tree edit distance allowing us to have strong theoretical justification while having an efficient procedure to deal with complex distance, such as the Zhang-Shasha one, which is a clear advantage in comparison of EM-based methods that quickly become intractable.

We experimentally showed on a music recognition task that this framework is able to build very accurate classifiers improving state-of-the-art results for this problem. This experiment was done in a multi-class setting

Table 4.2: Number of reasonable points used to learn a linear classifier between classical and children's music.

Song	# reasonable points
Toccata and fugue	4
Avemaria	1
Ode to joy	2
Bolero	2
Lohengrin, wedding march	5
Alouette	2
Oh! Susanna	4
Happy birthday	5
Twinkle twinkle little star	2
Jingle bells	2
Success (%)	94.29

showing that GESL can also deal with this context. Moreover, we illustrated that the produced models can be used to provide a semantic analysis of the knowledge learned from the data.

A perspective of this work would be to study other definitions of tree edit similarities. Indeed, GESL is based on a linear combination of the edit script operations plugged in an exponential but we could imagine other strategies tailored to the application at hand. Another interesting future work would be to adapt the similarity learning procedure directly to the multi-class setting instead of binary classification. Finally, the efficiency of this tree edit distance learning framework (both in terms of accuracy and running time) opens the door to an extensive use of tree edit distance in larger-scale applications such as natural language processing or XML data classification.

5

Tree Automata

Similarity computation is a difficult issue in music information retrieval tasks, because it tries to emulate the special ability that humans show for pattern recognition in general, and particularly in the presence of noisy data. A number of works have addressed the problem of what is the best representation for symbolic music in this context. The tree representation, using rhythm for defining the tree structure and pitch information for leaf and node labeling has proven to be effective in melodic similarity computation. One of the main drawbacks of this approach is that the tree comparison algorithms are of a high time complexity. In this chapter, stochastic k-testable tree-models are applied for computing the similarity between two melodies as a probability. The results are compared to those achieved by tree edit distances, showing that k-testable tree-models outperform other reference methods in both recognition rate and efficiency.

5.1 Introduction

In this chapter, the problem of comparing symbolically encoded (e.g. MIDI or MusicXML) musical works is addressed. For it, probabilistic k-testable tree models (Rico-Juan et al., 2005), a generalization of the k-gram models, are utilized. These models are easy to infer from samples and allow incremental updates. They can be used for data categorization if a model is inferred for each class and the new samples are assigned a probability by each model, taking a maximum likelihood decision. These models have been used in a number of applications, like structured data compression (Rico-Juan et al., 2005) or information extraction from structured documents (like HTML or XML, for example) (Kosalaa et al., 2006), but here they are applied to music data, focusing on the aforementioned inherent structured nature.

These models build a stochastic tree automata that parses a tree from the leaves (initial states) to its root (final states), assigning probabilities to the

transitions. In this process, the similarity between a melody and a reference model is computed as a probability.

5.2 Stochastic k-testable Tree Models

Stochastic models based on k-grams predict the probability of the next symbol in a sequence depending on the $k-1$ previous symbols. They have been extensively used in natural language modeling (Brown et al., 1990; Ney et al., 1995), speech recognition (Jelinek, 1998), and also in some music information retrieval tasks (Downie, 1999). From a theoretical point of view, k-gram models can be regarded as a probabilistic extension of locally testable languages (Zalcstein, 1972). Informally, a string language $\mathcal{L}$ is locally testable if every string w can be recognized as a string in $\mathcal{L}$ just by looking at all the substrings in w of length at most k, together with prefixes and suffixes of length strictly smaller than k to check near the string boundaries. These models are easy to learn and can be efficiently processed (García and Vidal, 1990; Yokomori, 1995).

Locally testable languages, in the case of trees, were described by Knuutila (García, 1993; Knuutila, 1993). In them, the concept of k-fork, f_k, plays the role of the substrings, and the k-root, r_k, and k-subtrees, s_k, play the role of prefixes and suffixes. For any $k > 0$, every k-fork contains a node and all its descendants lying at a depth smaller that k. The k-root of a tree is its shallowest k-fork and the k-subtrees are all the subtrees whose depth is smaller than k. These concepts are illustrated in Fig. 5.1 for the tree $t = a(a(a(ab))b)$. In this example, $r_2(t) = \{a(ab)\}$, $f_3(t) = \{a(a(a)b), a(a(ab))\}$, and $s_2(t) = \{a(ab), a, b\}$.

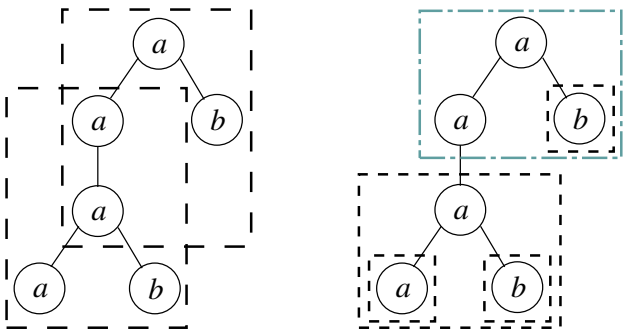


Figure 5.1: Left: set of 3-forks in $a(a(a(ab))b)$. Right: 2-root (in grey) and 2-subtrees (black dashed).

The stochastic k-testable models will be used to build models to classify melodies represented as trees. In the next section, we specify the notation and describe the models.

5.2.1 Trees and Tree Automata

Given an alphabet, that is, a finite set of symbols (also called labels) $\Sigma = \{\sigma_1, \cdots, \sigma_{|\Sigma|}\}$, we define the set T_Σ of Σ-trees as a Context free grammar $G = (\Sigma', \{T, F\}, T, R)$ where Σ and the left and right brackets are included in the alphabet Σ'. And R is a set of transition rules of the following type:

- $T \longrightarrow \sigma|\sigma(F) \quad \forall\ \sigma \in \Sigma$;
- $F \longrightarrow T|TF$.

The depth (*depth*) of a tree t is defined as:

$$depth(t) = \begin{cases} 0 & \text{if } t = \sigma \in \Sigma, \\ 1 + \max_{j=1}^{m}\{depth(t_j)\} & \text{if } t = \sigma(t_1 \cdots t_m) \in T_\Sigma - \Sigma \end{cases} \tag{5.1}$$

and the subset of trees is

$$sub(t) = \begin{cases} \{\sigma\} & \text{if } t = \sigma \in \Sigma, \\ \{t\} \cup \bigcup_n sub(t_n) & \text{if } t = \sigma(t_1 \cdots t_m) \in T_\Sigma - \Sigma. \end{cases} \tag{5.2}$$

For example, the depth of the tree $a(a(a(ab))b)$ that belongs to $T_{\{a,b\}}$ is 3 and its representation is shown in Fig. 5.1.

Finite Tree Automata

A Deterministic Finite Tree Automata (DFTA) is defined as a 4-tuple $A = (Q, \Sigma, \Delta, F)$, where

- $Q = \{q_1, \cdots, q_{|Q|}\}$ is a finite set of (unary) states;
- $\Sigma = \{\sigma_1, \cdots, \sigma_{|\Sigma|}\}$ is the alphabet;
- $F \subseteq Q$ is a set of final states;
- $\Delta = \{\delta_0, \delta_1, \cdots, \delta_M\}$ is a set of transition rules of the following type $\delta : \Sigma \times Q^m \longrightarrow Q$.

Let us note that there is no initial state in a DFTA, but, when $m = 0$, i.e. when the symbol is a constant symbol a, a transition rule is of the form $a \to q(a)$. Therefore, the transition rules for the constant symbols can be considered as the initial rules, i.e. for each leaf $\sigma \in \Sigma$, $\delta_0(\sigma)$ define an initial state. Otherwise, if $t = \sigma(t_1, t_2, \cdots, t_m)$ is a subtree with the label $\sigma \in \Sigma$ that expands m subtrees $t_1, t_2, \cdots, t_m$, the state $\delta(t) \in Q$ is $\delta_m(\sigma, \delta(t_1), \delta(t_2), \cdots, \delta(t_m))$. Therefore, $\delta(t)$ is recursively defined for $t = \delta(t_1, t_2, \cdots, t_m)$ as:

$$\delta(t) = \begin{cases} \delta_0(\sigma) & \text{if } m = 0 \\ \delta_m(\sigma, \delta(t_1), \delta(t_2), \cdots, \delta(t_m)) & \text{if } m > 0 \end{cases} \tag{5.3}$$

The language recognized by the automaton A is the subset T_Σ

$$L(A) = \{t \in T_\Sigma : \delta(t) \in F\} \tag{5.4}$$

For example, let $Q = \{q_1, q_2\}$, $\Sigma = \{a, b\}$ and Δ include the transitions $\delta_0(a) = q_1$, $\delta_0(b) = q_2$, $\delta_2(a, q_1, q_2) = q_2$ y $\delta_1(a, q_2) = q_1$, the result of processing with A the tree $t = a(a(a(ab))b)$ (Fig. 5.1) is $\delta(t) = \delta_2(a, \delta(a(a(ab))), \delta(b))$. Recursively we obtain, $\delta(a(a(ab))) = q_1$ and $\delta(b) = q_2$, therefore $\delta(t) = \delta(a, q_1, q_2) = q_2$. Lastly, for practical reasons, it is usual to consider automata in which unnecessary states are eliminated i.e. the transitions not defined lead to an absorption state.

5.2.2 Stochastic Tree Automata

Stochastic tree automata generate a probability distribution over the trees in T_Σ. A stochastic DTA incorporates a probability for every transition in the automaton, with the normalization that the probabilities of transitions leading to the same state $q \in Q$ must add up to one. In other words, there is a collection of functions $P = p_0, p_1, p_2, \ldots, p_M$ of the type $p_m : \Sigma \times Q^m \to [0, 1]$ such that they satisfy, for all $q \in Q$,

$$\sum_{\sigma \in \Sigma} \sum_{m=0}^{M} \sum_{\substack{q_1, \cdots, q_m \in Q : \\ \delta_m(\sigma, q_1, \cdots, q_m) = q}} p_m(\sigma, q_1, \cdots, q_m) = 1 \tag{5.5}$$

In addition to this probabilities, every *stochastic deterministic tree automaton* $A = (Q, V, \Delta, P, r)$ provides a function $r : Q \to [0, 1]$ which, for every $q \in Q$, gives the probability that a tree satisfies $\delta(t) = q$ and replaces, in the definition of the DTA, the subset of accepting states. Then,

the probability of a tree t in the language generated by the stochastic DTA A is given by the product of the probabilities of all the transitions used when t is processed by A, times(multiplied) $r(\delta(t))$:

$$p(t|A) = r(\delta(t))\pi(t). \tag{5.6}$$

with $\pi(t)$ recursively given by

$$\pi(t) = \begin{cases} p_0(\sigma) & \text{if } m = 0 \\ p_m(\sigma, \delta(t_1), \delta(t_2), \cdots, \delta(t_m))\pi(t_1)\pi(t_2)\cdots\pi(t_m) & \text{if } m > 0 \end{cases} \tag{5.7}$$

The equations (5.6) and (5.7) define a probability distribution $p(t|A)$ which is consistent if

$$\sum_{t \in T_\Sigma} p(t|A) = 1 \tag{5.8}$$

As put forward in (Chaudhuri, 1983) and (Sánchez and Benedí, 1997), context free grammars whose probabilities are estimated from random samples are always consistent. It is easy to show (Sakakibara, 1992) that the language recognized by a DTA can be generated also with a regular tree-grammar. In the following, the probabilities of the DTA will be extracted from random samples and, therefore, consistency is always preserved.

5.2.3 Locally Testable Tree Languages

In this section, the k-testable tree languages and its inference methods are defined following the ideas appeared in (Knuutila, 1993) y (García and Vidal, 1990). For that, define the sets of k-forks, k-subtrees and k-root of a tree is needed. These sets are generated from tree fragments are obtained when a tree is observed through a window of k size, in other words, with a maximum depth of k. In the case of the strings k-grams models, we have to specify what happens at the begin and the end of each string. In the same way, the root (k-root) and the leaves (k-subtrees) have to be processed explicitly.

For all $k > 0$ and for all trees $t = \sigma(t_1 \ldots t_m) \in T_\Sigma$. the k-root of t is a tree in T_Σ defined as

$$r_k(t) = \begin{cases} \sigma(r_{k-1}(t_1), r_{k-1}(t_2), \cdots, r_{k-1}(t_m)) & \text{if } k > 1 \\ \sigma & \text{if } k = 1 \end{cases} \tag{5.9}$$

Note that in case $m = 0$, that is $t = \sigma \in \Sigma$, then $r_k(\sigma) = \sigma$. On the other hand, the set $f_k(t)$ of k-forks and the set $s_k(t)$ of k-subtrees are defined as follows:

$$f_k(t) = \bigcup_{j=1}^{m} f_k(t_j) \bigcup \begin{cases} r_k(t) & \text{if depth}(t) \geq k-1 \\ \emptyset & \text{in other case} \end{cases} \tag{5.10}$$

$$s_k(t) = \bigcup_{j=1}^{m} s_k(t_j) \bigcup \begin{cases} t & \text{if depth}(t) \leq k-1 \\ \emptyset & \text{in other case} \end{cases} \tag{5.11}$$

In the particular case $t = \sigma \in \Sigma$, then $s_k(t) = f_1(t) = \sigma$ y $f_k(t) = \emptyset$ for all $k > 1$. For instance, if $t = a(a(a(ab))b)$ then one gets $r_2(t) = \{a(ab)\}$, $f_3(t) = \{a(a(a)b), a(a(ab))\}$, and $s_2(t) = \{a(ab), a, b\}$. Note that these definitions coincide with those in (Knuutila, 1993)except for the meaning of k.

A tree language T is an strictly k-testable language (with $k \geq 2$) if there exist finite subsets $\mathcal{R}, \mathcal{F}, \mathcal{S} \subseteq T_\Sigma$ such that

$$t \in T \Leftrightarrow r_{k-1}(t) \subseteq \mathcal{R} \wedge f_k(t) \subseteq \mathcal{F} \wedge s_{k-1}(t) \subseteq \mathcal{S}. \tag{5.12}$$

Equation (5.12) show that to identify a k-testable tree language T, three sets of the tree have to be identify for each tree: the root, the forks with depth k and the subtrees with depth less than or equal to $k-1$. The example in figure Fig. 5.1, if t belongs to the T language, $\mathcal{R}$ have to include $a(ab)$, $\mathcal{F}$ have to include $a(a(a)b)$ and $a(a(ab))$ and $\mathcal{S}$ have to include $a(ab)$, a and b.

In such a case, it is straightforward (García, 1993; Knuutila, 1993) to build a DTA $A = (Q, \Sigma, \Delta, F)$ that recognizes T. For this purpose, it suffices:

$$\begin{aligned} &Q = \mathcal{R} \cup r_{k-1}(\mathcal{F}) \cup \mathcal{S}; \\ &F = \mathcal{R}; \\ &\delta_m(\sigma, t_1, \cdots, t_m) = \sigma(t_1, \cdots, t_m) \forall \sigma(t_1, \cdots, t_m) \in \mathcal{S}; \\ &\delta_m(\sigma, t_1, \cdots, t_m) = r_{k-1}(\sigma(t_1, \cdots, t_m)) \forall \sigma(t_1, \cdots, t_m) \in \mathcal{F}; \end{aligned} \tag{5.13}$$

If one assumes that the tree language L is k-testable, the DTA recognizing L can be identified from positive samples (García, 1993; Knuutila, 1993), that is, from sets made of examples of trees in the language. Given a positive sample Ω, the procedure to obtain the DTA essentially builds the automaton A using $r_{k-1}(\Omega)$, $f_k(\Omega)$ y $s_{k-1}(\Omega)$ instead of $\mathcal{R}$, $\mathcal{F}$ and $\mathcal{S}$ respectively in the above definitions for Q, F y Δ in equation 5.13.

For example, given the tree $t = a(a(a(ab))b)$, $r_2(t)$, $f_3(t)$ and $s_2(t)$ processed previously in Fig. 5.1, as $r_2(f_3(t)) = \{a(ab), a(a)\}$, the corresponding DTA will be:

$$
\begin{aligned}
&Q = \{a(ab), a(a), a, b\};\\
&F = \{a(ab)\};\\
&\delta_0(a) = a;\\
&\delta_0(b) = b;\\
&\delta_2(a, a, b) = a(ab);\\
&\delta_2(a, a(a), b) = a(ab);\\
&\delta_1(a, a(ab)) = a(a);
\end{aligned}
\tag{5.14}
$$

In this way, the tree t is recognized by the DTA, i.e., $t \in L(A)$.

Throughout this work we will use the name of the rules to refer to the trees which define transitions in the automata. In other words, each tree t that belongs $f_k(\Omega) \bigcup s_{k-1}(\Omega)$ will be a rule and the trees $r_{k-1}(t) \in Q$ which generate the rules will be the states. In the previous example, the states are $\{a(ab), a(a), a, b\}$ and the rules $\{a, b, a(ab), a(a(a)b), a(a(ab))\}$ associated to the states are $\{a, b, a(ab), a(ab), a(a)\}$.

Estimating Transition Probabilities

A stochastic sample $\Omega = \{\tau_1, \tau_2, \cdots, \tau_{|S|}\}$ consists of a sequence of trees generated according to a given probability distribution. If our model is a stochastic DTA, the distribution is $p(t|A)$ as given by equations (5.6) y (5.7). Again, the assumption that the underlying transition scheme (that is, the states Q and the collection of transition functions Δ) correspond to a k-testable allows one to infer a stochastic DTA from a sample in a simple way.

For this purpose, one should note that the likelihood of the stochastic sample Ω is given by

$$\prod_{i=1}^{n} p(\tau_i|A) \tag{5.15}$$

is maximized (Ney et al., 1995) if the automaton A assigns to every tree τ in the sample a probability equal to the relative frequency of τ in Ω. In other words, every transition in Δ is assigned a probability which coincides with the relative number of times the rule is used when the trees in the sample are parsed. Summarizing, given a stochastic sample $\Omega = \{\tau_1, \tau_2, \cdots, \tau_{|S|}\}$, the set of states is

$$Q = r_{k-1}(\Omega) \bigcup r_{k-1}(f_k(\Omega)) \bigcup s_{k-1}(\Omega); \tag{5.16}$$

the subset of accepting states is

$$F = r_{k-1}(\Omega); \tag{5.17}$$

the probabilities $r(t)$ are estimated from Ω as

$$r(t) = \frac{1}{|\Omega|} C_r^{[k-1]}(t, \Omega), \tag{5.18}$$

where

$$C_r^{[k-1]}(t, \Omega) = \sum_{i=1}^{|\Omega|} C_r^{[k-1]}(t, \tau_i)$$

and

$$C_r^{[k-1]}(t, \tau) = \begin{cases} 1 & \text{si } r_{k-1}(\tau) = t \\ 0 & \text{en othecase} \end{cases}$$

and, finally, the transition probabilities P are estimated as

$$p_m(\sigma, t_1, \cdots, t_m) = \frac{C^{[k]}(\sigma(t_1, \cdots, t_m), \Omega)}{\sum_{i=1}^{|\Omega|} C^{[k-1]}(r_{k-1}(\sigma(t_1, \cdots, t_m)), \Omega)} \tag{5.19}$$

where

$$C^{[k-1]}(t, \Omega) = \sum_{i=1}^{|\Omega|} C^{[k-1]}(t, \tau_i)$$

and $C^{[k-1]}(t, \tau)$ counts the number of k-forks or $(k-1)$-subtrees to t in τ.

It is useful to store the above probabilities r and p as the quotient of two terms, as given by equations (5.18) y (5.19). In this way, if a new sample Ω' is provided, the automaton A can be easily updated to account for the additional information. For this incremental update, it suffices to increment each term in the equations with the sums obtained for the new sample.

Approximating Stochastic DTA by k-testable Automata

From the construction, it is obvious that all stochastic k-testable languages can be generated by a stochastic DTA. However, as it is also the case with strings languages (Stolcke and Segal, 1994), the reciprocal is not always true. The best approximate k-testable model can be obtained in the following way, based upon the results in (Calera-rubio and Carrasco, 1998). Assume that we are given a stochastic DTA $A = (Q, \Sigma, \Delta, P, r)$. For any value of k, we obtain a k-testable stochastic DTA $A' = (Q', \Sigma', \Delta', P', r')$ whose probabilities are given by

$$r'(j) = \sum_{i \in Q} r(i)\eta_{ij} \tag{5.20}$$

for all $j \in Q'$ and

$$p'_m(\sigma, j_1, \cdots, j_m) = \frac{\sum_{i_1,\cdots,i_m \in Q} C_{\delta(\sigma,i_1,\cdots,i_m)} p_m(\sigma, i_1, \cdots, i_m)\eta_{ij}}{\sum_{i \in Q} C_i} \tag{5.21}$$

where C_i is the expected number of nodes of type i in a tree and η_{ij} represents the probability that a node i expands as a subtree t such that $r_{k-1}(t) = j$. All these coefficients can be easily computed (Calera-rubio and Carrasco, 1998) using iterative procedures. In particular, C_i is given by

$$C_i^{[n+1]} = r(i) + \sum_{j \in Q} \Lambda_{ij} C_j^{[n]} \tag{5.22}$$

with $C_i^{[0]} = 0$ and

$$\Lambda_{ij} = \sum_{m=1}^{M} \sum_{\sigma \in \Sigma} \sum_{\substack{j_1, \cdots, j_m \in Q : \\ \delta_m(\sigma, j_1, \cdots, j_m) = j}} p_m(\sigma, j_1, \cdots, j_m)(\delta_{ij_1} + \cdots + \delta_{ij_m}) \tag{5.23}$$

The coefficients $\eta_{ij}^{[n]}$ are computed as

$$\eta_{ij}^{[n+1]} = \sum_{m=0}^{M} \sum_{\sigma \in \Sigma} \sum_{\substack{i_1, \cdots, i_m \in Q : \\ \delta_m(\sigma, i_1, \cdots, i_m) = i}} \sum_{\substack{j_1, \cdots, j_m \in Q' : \\ \delta'_m(\sigma, j_1, \cdots, j_m) = j}} p_m(\sigma, i_1, \cdots, i_m)(\eta_{i_1 j_1}^{[n]} + \cdots + \eta_{i_m j_m}^{[n]}) \tag{5.24}$$

starting with $\eta_{ij}^{[0]} = 0$ (note that the terms with $m = 0$ do not necessarily be null). Obviously, the cross entropy between the exact model A and the approximate one A' can also be computed following the method described in (Calera-rubio and Carrasco, 1998).

5.3 Classification

In a stochastic classification task, a sample is assigned to the class that maximizes the probability of generating it. For a new tree to be classified in

a particular class of trees, we need to infer a PDTA for each class, C_j, from well classified trees.

5.3.1 Introduction

The general approach to classify a new sample represented as a tree consists in the inference of a DTA for each class from well classified samples. Thus, a new sample is classified in the class that maximizes the likelihood of the new sample for each class (Duda and Hart, 1973). For that, when the DTA is inferred, the conditional probability of a new tree is computed as the product of the probabilities for each transition used to parse the tree.

The problem occurs when the metric associated to the new tree do not have suffice information. How we said above, the conditional probability is calculated multiplying the probabilities of the transitions used to parse the tree. Suppose that some of these transitions of the tree is not in the PDTA that is processing the new tree. In this case, the probability assigned to the new tree by the model is zero even when only one of the transitions are not in the automaton and the others are. In that way, although the model was the more indicated (without the null transition), the winner class pass to be another because of the model will be a null probability.

A worse case occurs when we try to classify a new tree and this has some unseen transition in each model. In this case, all models will be a null probability and the assignation of a winner class will be random and not reliable. To avoid these cases is needed to smooth the models in order to do not assign null probabilities when unseen transitions appear. Next section we describe some smoothing methods for the tree case.

5.3.2 Smoothing Methods

Model Interpolation

The inference approach described previously allow to obtain different stochastic models $M^{[k]}$, one for each value of k from 2 to K. Intuitively, a model with a big k is more probable that assigns a null probability to a new tree due to a lot of parameters could not be estimated from the sample. Actually, a 2-testable model could predict a null probability in some cases.

Two classical smoothing techniques are linear interpolation and backing-off (Ney et al., 1995). Smoothing through linear interpolation is performed by computing the probability of events as a weighted average of the probabilities given by different model sizes. The probability is calculated as a linear

combination of each models (with fixed coefficients), even the base model, $M^{[1]}$, which never assign null probabilities.

If the probability of a tree is calculated as a linear combination of each models:

$$p(\tau|A) = \sum_{k=1}^{K} \alpha_k p(\tau|M^{[k]}) \tag{5.25}$$

with the restriction $\sum_{k=1}^{K} \alpha_k = 1$, we are sure, if $\alpha_1 > 0$, that $p(\tau|A) > 0$ for every tree τ.

Therefore, it is important to have a base model $M_{[1]}$ that never assign null probabilities in other case the interpolation will assign null probabilities in unseen transitions.

But if the higher k models return a zero probability, the target tree will be classified only with the more general and more ambiguous model discarding the entire, more specific, k-sized model. This fact harms the classification precision.

Due to the nature of the data and the tests carried out, the interpolation approach was rejected from the beginning since the different models provided null probabilities in about the totality of de sample so that the use of this approach have been similar to the use of a unique base model $M^{[1]}$.

Tree languages multilevel smoothing (*Backing-off*)

Backing-off methods have been extensively studied for string models (Ney et al., 1995). The underlying idea is to discount some probability mass to the seen events and distribute it among the unseen events. For it, models with $1 \leq k \leq K$ are needed (K is the more specific model). The smaller the k value, the more general the model is. The aim is always to compute the probability of a transition with the K model. If it does not have the needed transition then the $k-1$ model is utilized. If necessary, this process is repeated until the $k = 1$ model is used. This is a base model that never assigns null probabilities and therefore is able to recognize any tree through its component nodes. Note this approach is a *transition-based backing-off* that calculate the probability of a transition with the more specific model possible. Thus solve the existing problem in a *model-based backing-off* approach when a single unseen transition can cause the model k returns a zero probability even if the rest of transitions have been seen.

Following an standard approach, the back-off schema uses a probability distribution p_A with another more general, p_B as follows:

$$p(x) = \begin{cases} p_A(x)(1-\lambda(x)) & \text{if } p_A > 0 \ , \\ \frac{\Lambda}{F} p_B(x) & \text{other case.} \end{cases} \tag{5.26}$$

The p_A distribution is decreased with a discount function λ such that $0 < \lambda(x) < 1$ and yield the discount term

$$\Lambda = \sum_{x:p_A(x)>0} \lambda(x) p_A(x) \tag{5.27}$$

which will be distributed in all the unseen events. The factor F is needed to keep the normalization:

$$F = \sum_{x:p_A(x)=0} p_B(x) \tag{5.28}$$

In our case, as explained in Section 3.3, a melody is formed by a given number of bars and each bar is represented by a tree (see Fig. 3.3). Therefore we are able to compute the probability of each bar tree to belong to a particular class by parsing it with the PDTA of that class. If we compute the probability of each bar to belong to a class, those probabilities can be combined to give a likelihood for the whole song, $l(M|C_j)$, and this procedure can be performed for all the classes. For the combination of bar probabilities three different strategies have been used that are explained below.

A melody M is represented as a forest $\sigma(t_1, t_2, ..., t_{|M|})$ which have a common root σ and a number of trees, $|M|$, equal to the number of bars in the melody. Based on the equations (5.6) and (5.7), the probability, adapted to the k-testable languages, of a melody M in the model $A^{[k]}$ can be defined as:

$$p(M|A^{[k]}) = \rho^{[k]}(r_{k-1}(M)) \prod_{i=1}^{|M|} \pi^{[k]}(t_i) \tag{5.29}$$

where each tree corresponding to the different bars is $t = \sigma(t_1, t_2, ..., t_m)$ and $\pi^{[k]}(t)$ is recursively defined as

$$\pi^{[k]}(t) = \begin{cases} p_0^{[k]}(\sigma) & \text{if } m = 0 \\ p_m^{[k]}(\sigma, r_{k-1}(t_1), r_{k-1}(t_2), \cdots, r_{k-1}(t_m)) \prod_{i=1}^{|m|} \pi^{[k]}(t_i) & \text{if } m > 0 \end{cases} \tag{5.30}$$

Following, the base model $k = 1$, which can be recognize any tree, is defined. This model defines two probability distributions, the first for the

number of children m of a node labelled as σ , $p_\sigma^{[1]}(m)$, and the second, for the label $p^{[1]}(\sigma)$. In the first case $p_\sigma^{[1]}(m)$ is defined as

$$p_\sigma^{[1]}(m) = \begin{cases} \frac{E_\sigma(m)}{C^{[1]}(\sigma)}(1-\lambda_\sigma^{[1]}(m)) & \text{if } E_\sigma(m) > 0 \ , \\ \frac{\Lambda_\sigma^{[1]}}{F_\sigma^{[1]}} P_\sigma(m) & \text{if } E_\sigma(m) = 0 \wedge C^{[1]}(\sigma) > 0 \ , \\ P(m) & \text{other case.} \end{cases} \tag{5.31}$$

where $E_\sigma(m)$ is the number of times the label σ appears with m children. In our case, the tree aridity is bounded, each tree only can have $m = 0, 2$, or 3 children. $C^{[1]}(\sigma)$ is is the number of times the label σ appears in the training sample. $\lambda_\sigma^{[1]}$ is a discount function which is strictly positive and generates, for each σ, a global discount factor.

$$\Lambda_\sigma^{[1]} = \frac{1}{C^{[1]}(\sigma)} \sum_m E_\sigma(m) \lambda_\sigma^{[1]}(m), \tag{5.32}$$

which is normalized as

$$F_\sigma^{[1]} = 1 - \sum_{n:E_\sigma(n)>0} P_\sigma(n). \tag{5.33}$$

$P_\sigma(m)$ is the priori probability that a label σ appear with m descendants. This probability can not be zero. For that, it is considered that each different label has been seen one time at least with 0, 2 or 3 children. $P(m)$ is the priori probability of the nodes with m children.

These probabilities are used here to calculate the base model $k = 1$ instead the Poisson distributions used in (Rico-Juan et al., 2005) because in this work the aridity is bounded.

Secondly, the probability $p^{[1]}(\sigma)$ of a node is labelled with σ without any context is defined:

$$p^{[1]}(\sigma) = \begin{cases} \frac{C^{[1]}(\sigma)}{N}(1-\lambda^{[1]}(\sigma)) & \text{if } C^{[1]}(\sigma) > 0 \ , \\ \frac{\Lambda^{[1]}}{F^{[1]}} & \text{other case} \end{cases} \tag{5.34}$$

where $\lambda^{[1]}(\sigma)$ is a function strictly positive, $N = \sum_a C^{[1]}(a)$ is the number of nodes in the sample, $F^{[1]}$ is the normalization factor calculated as the number of different symbols in the alphabet which do not have been found in the training sample and

$$\Lambda^{[1]} = \frac{1}{N} \sum_a C^{[1]}(a) \lambda^{[1]}(a). \tag{5.35}$$

Should be consider that if all symbols of de alphabet have been found in the training sample, this discount factor is not needed and then, the calculation of the probabilities will be done as follows

$$p^{[1]}(\sigma) = \frac{C^{[1]}(\sigma)}{N} \tag{5.36}$$

Whit these ingredients, we are ready to define the schema *Backing-off* for the basic probabilities with $k = 2$. Given the tree $t = \sigma(a_1...a_m)$, is processed as follows:

$$p_m^{[2]}(\sigma, a_1, ..., a_m) = \begin{cases} \frac{C^{[2]}(t)}{C^{[1]}(\sigma)}(1 - \lambda^{[2]}(t)) & \text{if } C^{[2]}(t) > 0 \ , \\ \frac{\Lambda^{[2]}(\sigma)}{F^{[2]}(\sigma)} p_\sigma^{[1]}(m) \times \prod_{i=1}^m p^{[1]}(a_i) & \text{if } C^{[2]}(t) = 0 \wedge C^{[1]}(\sigma) > 0 \ , \\ p_\sigma^{[1]}(m) \times \prod_{i=1}^m p^{[1]}(a_i) & \text{other case} \end{cases} \tag{5.37}$$

where

$$\Lambda^{[2]}(\sigma) = \sum_{u: C^{[2]}(u) > 0 \wedge r_1(u) = \sigma} \frac{C^{[2]}(u)}{C^{[1]}(\sigma)} \lambda^{[2]}(u) \tag{5.38}$$

and the normalization factor is

$$F^{[2]}(\sigma) = 1 - p_\sigma^{[1]} - \sum_{m>0: E_\sigma(m) > 0} p_\sigma^{[1]}(m) \times \sum_{\substack{a_1, ..., a_m : \\ C^{[2]}(\sigma(a_1...a_m)) > 0}} \prod_{i=1}^m p^{[1]}(a_i). \tag{5.39}$$

For the case $k > 2$, $p_0^{[k]}(\sigma) = 1$.

Para el caso $k > 2$, $p_0^{[k]}(\sigma) = 1$. However, the probabilities of the type $p_m^{[k]}$ with $m > 0$, given $t = \sigma(t_1, \cdots, t_m)$ with $t_i = a_i(s_{i1}...s_{im_i})$ are defined as

$$p_m^{[k]}(\sigma, t_1, ..., t_m) = \begin{cases} \frac{C^{[k]}(t)}{C^{[k-1]}(r_{k-1}(t))} \times (1 - \lambda^{[k]}(t)) & \text{if } C^{[k]}(t) > 0 \ , \\ \frac{\Lambda^{[k]}(r_{k-1}(t))}{F^{[k]}(r_{k-1}(t))} \times \prod_{i=1}^m p_{m_i}^{[k-1]}(a_i, s_{i1}, ..., s_{im_i}) & \text{if } C^{[k]}(t) = 0 \wedge \\ & C^{[k-1]}(r_{k-1}(t)) > 0 \ , \\ \prod_{i=1}^m p_{m_i}^{[k-1]}(a_i, s_{i1}, ..., s_{im_i}) & \text{other case} \end{cases} \tag{5.40}$$

where

$$\Lambda^{[k]}(t) = \frac{1}{C^{[k-1]}(t)} \sum_{u:C^{[k]}(u)>0 \wedge r_{k-1}(u)=t} C^{[k]}(u)\lambda^{[k]}(u) \tag{5.41}$$

and the normalization factor is

$$F^{[k]}(\lambda(t_1...t_m)) = 1 - \sum_{\substack{u_1,...,u_m : \\ C^{[k]}(\sigma(u_1...u_m))>0 \wedge r_{k-2}(u_i)=t_i}} \prod_{i=1}^{m} p_{m_i}^{[k-1]}(a_i, v_{i1}, ..., v_{im_i}). \tag{5.42}$$

Note that $p_0^{[k]} = 1 \ \forall t$.

The third case in equations (5.37) and (5.40) was not taken into account in theoretical work realized by Rico and coworkers (Rico-Juan et al., 2005) because, when was applied to the natural language processing, the coverage of de model $k-1$ was complete. This is not the case of de musical data, due to the specific features and the size of the sample used. Therefore, for the case where the model $k-1$ do not have been saw a transition a similar schema of n-grams used in (Katz, 1987) is applied.

Finally, the root probabilities $\rho^{[k]}$ have to be defined. A melody M is represented as a forest $\sigma(t_1, t_2, ..., t_{|M|})$ which have a common root and a number of trees, $|M|$, equal to the number of bars in the melody. Each tree represent the notes in a bar (see Fig. 3.3). Therefore, the number of children of the common root is not bounded in the same way a melody can have an infinite number of bars.

To carry out the calculus of $\rho^{[k]}$ we only consider the roots of the model $k = 2$. In this way only a root label of the forests is used. This arrangement is considered to avoid that the probability of a given melody belongs to a class being dependent of the number of bars that the melody have (its length). In the case of we need use roots with a great k, in other kind of applications, we have to use Poisson estimators due to the number of children of the root is not bounded. Therefore, $\rho^{[2]}(\sigma)$ is defined as:

$$\rho^{[2]}(\sigma) = \begin{cases} \frac{D^{[2]}(\sigma)}{|\Omega|}(1 - \theta^{[2]}(\sigma)) & \text{if } D^{[2]}(\sigma) > 0 \ , \\ \frac{\Theta^{[2]}}{G^{[2]}} p_\sigma^{[1]}(0) & \text{other case} \end{cases} \tag{5.43}$$

$\theta^{[2]}(\sigma)$ are strictly positive functions. The discount factor Θ^2 is

$$\Theta^{[2]} = \frac{1}{|\Omega|} \sum_{a:D^{[2]}(a)>0} D^{[2]}(a)\theta^{[2]}(a) \tag{5.44}$$

and its corresponding normalization

$$G^{[2]} = 1 - \sum_{a:D^{[2]}(a)>0} p^{[1]}(a). \tag{5.45}$$

$D^{[2]}(\sigma)$ is the number of times that the label σ appear as a root. In the same way that happens in equation (5.36), if every labels have found as roots in the training set, the same procedure is carried out.

Backing-off uses a discount parameter $\lambda^{[k]}(t)$ for transitions probability and other $\theta^{[k]}(t)$ for the roots probability (accepting state) of the tree. To estimate these parameters we use the Good-Turing Frequency Estimation (Gale, 1994).

When the probabilities have been estimated with the inferred models for each class $A_{C_j}^{[k]}$, a melody M is classified in the class $\hat{C}$ that maximizes the probability

$$\hat{C} = \arg\max_j l(M|A_{C_j}^{[k]}) \tag{5.46}$$

where $l(M|A_{C_j}^{[k]})$ is processed in the same way that inequality (5.29).

Bar Strategies

Once the PDTAs for the different classes have been inferred and the probabilities estimated, a melody M is classified in the class $\hat{C}$ that maximizes the likelihood of the melody for each class

$$\hat{C} = \arg\max_j \ l(M|C_j) \tag{5.47}$$

that needs to be computed as described next.

As explained in Section 3.3, a melody is formed by a given number of bars and each bar is represented by a tree (see Fig. 3.3). Therefore we are able to compute the probability of each bar tree to belong to a particular class by parsing it with the PDTA of that class. If we compute the probability of each bar to belong to a class, those probabilities can be combined to give a likelihood for the whole song, $l(M|C_j)$, and this procedure can be performed for all the classes. For the combination of bar probabilities three different strategies have been used that are explained below.

In a first strategy called voting, a vote for each bar tree of the melody is given to the class whose probability is the highest for that tree. Thus, using this strategy to determine the class of the whole melody, the winning class will receive a greater number of votes. Then the likelihood can be computed as

$$l(M|C_j) = \sum_{i=1}^{|M|} l(t_i|C_j) \tag{5.48}$$

where

$$l(t_i|C_j) = \begin{cases} 1 & \text{if } C_j = C_k,\ \hat{C} = \arg\max_k\ p(t_i|C_k) \\ 0 & \text{if } C_j \neq C_k,\ \hat{C} = \arg\max_k\ p(t_i|C_k) \end{cases} \tag{5.49}$$

The second strategy (called arithmetic) to determine the most likely class for a melody is to calculate the arithmetic mean of the probabilities for all the bars that make up the melody. For that, the probabilities of all the bars for a particular class are summed up.

$$l(M|C_j) = \sum_{i=1}^{|M|} p(t_i|C_j) \tag{5.50}$$

Next, the result should be divided by the number of bars $|M|$ in the melody, but this operation is not needed for the classification because, given a particular melody problem, $|M|$ is the same for all classes. One drawback of this approach is that the average is sensitive to outliers. A third possible option (called geometric) is compute the geometric mean. The geometric mean is less sensitive to outliers than the arithmetic one. To compute it we only have to multiply all the probabilities.

$$l(M|C_j) = \prod_{i=1}^{|M|} p(t_i|C_j) \tag{5.51}$$

The $|M|$-th root of the resulting product is not needed for classification due to the same reasons above.

Eventually, we incorporate the probability that the root σ of the forest belongs to the class. This probability is calculated as the relative frequency of σ appears in the root of the forest in training set.

Dynamic time model

In the classification model described above, each bar is parsed independently and no temporal constrains are applied. Therefore, the different bars of a melody could be shuffled and the system would attribute the same likelihood to the resulting melody. In order to avoid this situation, a time model is introduced. For that, we use a k-gram framework applied to the roots of the bar parsing trees, in such a way that we only need the conditional probability

of the i-th bar (represented by the $(k-1)$-root, which is the accepting state of its parsing), given the $(i-1)$-th bar $(k-1)$-root, i.e. $p(\delta(t_i)|\delta(t_{i-1}), C_j)$ (see Fig. 5.2). These probabilities are also conditioned by the different classes, C_j, introducing this way more discrimination power. Note that these roots are summarizing the melodic information in each bar, so the different variations are somehow hidden by this coarse-level descriptor.

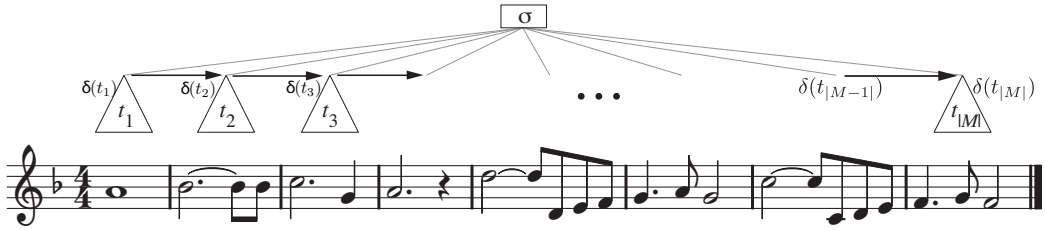


Figure 5.2: Representation of a $|M|$-bar melody with the temporal model depicted.

Using this information, the equations (5.49), (5.50), and (5.51) are modified, substituting a new $p'(t_i|C_j)$ for $p(t_i|C_j)$ that is computed as

$$p'(t_i|C_j) = p(\delta(t_i)|\delta(t_{i-1}), C_j) \cdot p(t_i|C_j) \tag{5.52}$$

5.4 Results

In our experiments, we have tried to identify a target melody using a set of different variations played by musicians for training the models for the different classes. We have tested our methodology using three different corpora.

The first corpus, named *Pascal* database consisting of a set of 420 monophonic 8-12 bar incipits of 20 worldwide well known tunes of different musical genres (see 3.4.1 for details).

Two other corpora (*Essen-Kinder* and *Essen-Lied*) have been used for testing the proposed method in order to test its performance on other kind of variations automatically generated (see 3.4.2 for details).

Different values for the maximum value of the model size, $K = \{2, 3, 4\}$, were utilized. Note that K is the largest (more specific) model used. This means that when $K = 2$, both $k = 2$ and $k = 1$ models are used; when $K = 3$, the models $k = 3$, $k = 2$, and $k = 1$ are used. They are combined with a *transition-based backing-off* scheme, as explained in Section 5.3.2. Also, the values for the discount parameters where established heuristically:

for the tree roots, $\theta^{[k]}(t) = 5 \cdot 10^{-7}$ for all models and trees, and for the rules, $\lambda^{[k]}(t) = 0.35$ for all the models and trees.

A 3-fold cross-validation scheme was carried out to perform the experiments, where 2/3 of the databases were used for training and 1/3 for test, obtaining average success rates and dispersions.

In Table 5.1 the results using the different approaches for combining the probabilities assigned to each bar explained in Section 5.3 are shown. In the first part (top) of the table, the results of the three approaches using only the probabilities of transitions are presented. Note that the results of votes and arithmetic mean approaches are much worse than using the geometric mean. These results are explained because the geometric mean is less sensitive to outliers, so variations that may have introduced exotic values when parsing a particular bar are filtered out more effectively. Thus, if most of the bars of the melody have a high probability for a given class, the melody is assigned to that class, although a few bars might have a very low probability.

Table 5.1: Success rates (%) with the different approaches used using *Pascal* corpus.

Approach	$K = 2$	$K = 3$	$K = 4$
Votes	47 ± 3	53 ± 4	48 ± 2
Arithmetic	38 ± 2	43 ± 2	40.7 ± 0.8
Geometric	84.3 ± 0.8	90.7 ± 0.8	81 ± 2
with tree roots			
Votes	52 ± 1	65 ± 2	72 ± 1
Arithmetic	41.2 ± 0.3	56 ± 2	64 ± 1
Geometric	86.5 ± 0.2	92.65 ± 0.03	93.1 ± 0.9
with forest root			
Arithmetic	54 ± 3	67 ± 3	71 ± 3
Geometric	88.2 ± 0.4	92.6 ± 0.4	94 ± 1
Time Model	91.9 ± 0.3	95.1 ± 0.2	96.1 ± 0.6

In the middle part of the table, the results adding the probabilities of the bar tree roots are presented. In this occasion, note how the use of this probability improves the success rates, especially when K is high. This is because for high K values the model is more specific and gives a much higher probability to the roots that have been seen during the training phase. This property is possible due to the use of a *transition-based backing-off*, because using a *model-based backing-off*, the more specific models would have zero

probability and the classification would be based solely on the $k = 1$ or $k = 2$ models.

At the bottom part of the table, the results incorporating the probability of the forest root are shown. Note that this strategy does not improve performance significantly with respect to the use of the bar tree roots. Note that in this situation, the voting approach has no sense.

The last line in table 5.1 (Time Model) shows the results using the temporal information. Note how using this information the results have improved significantly. These results point to the importance of the temporal information in the model.

In Table. 5.2 the results using the proposed maximum likelihood technique have been compared to those obtained for the same data and conditions tested in (Habrard et al., 2008), where for each target melody, classical tree edit distances (Selkow, 1977) to all the prototypes were computed and the nearest neighbor rule was applied. The edit distance was computed with insertion, deletion, and substitution costs set to 1.

Note that stochastic k-testable methods significantly improved for all K the results of the classical tree edit distance.

Table 5.2: Success rates (%) and comparison with the tree edit distance (*Pascal*) using the heuristically established parameter values.

	Tree edit dist.	$K = 2$	$K = 3$	$K = 4$
Without time model	82.0 ± 0.2	88.2 ± 0.4	92.6 ± 0.4	94 ± 1
Using time model		91.9 ± 0.3	95.1 ± 0.2	96.1 ± 0.6

In the Table. 5.3 the results using the Good-Turing method for estimating the discount parameters for the proposed scheme is compared to the results of the stochastic tree edit distance method used in (Habrard et al., 2008). The estimation with Good-Turing improved the success rate of the stochastic tree edit distance in all cases but for $K = 2$ without using the time model although they were not better than those obtained with the heuristic estimation.

Fig. 5.3 shows the success rates taking into account the first n most probable classes for different values of K. Note that, with the first 3 classes, the success rates reach around 97 % with all K values. Therefore, we can say that the correct song was usually among the first solutions. Note that, in the case where we took only the first class of the ranking, the $K = 2$ model worked worse than the others, because it is a more general model and it can not differentiate properly between the first two classes (this effect can be seen also in tables 5.1, 5.2, and 5.3).

Table 5.3: Success rates (%) and comparison with the stochastic edit distance (*Pascal*) using Good-Turing estimation for parameter values.

	Stochastic	$K = 2$	$K = 3$	$K = 4$
Without time model	90.7 ± 0.6	85.8 ± 0.9	92.9 ± 0.4	91 ± 1
Using time model		91.9 ± 0.2	94.6 ± 0.9	94.4 ± 0.9

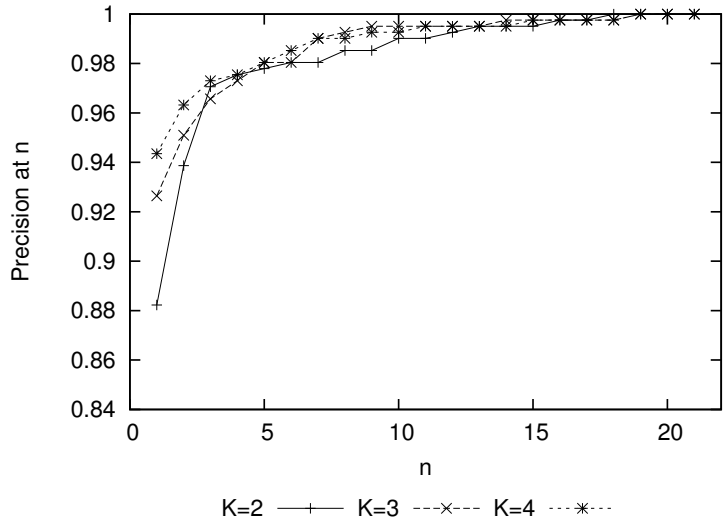


Figure 5.3: Success rates as a function of the number of classes retrieved as the most probable ones (*Pascal*).

In Fig. 5.4 the evolution of the success rate for the $K = 3$ model is displayed with respect to the number of samples used for training (the other models obtained similar results). As shown in the graph, it took only about 3 or 4 training samples to get an acceptable success rate (between 85 and 90 %). This shows that the system would be feasible even if very few training data were available. This is a very common situation in music information retrieval tasks, where the amount of labelled data use to be very limited.

Table 5.4 shows the success rate for the *Essen-Kinder* and *Essen-Lied* corpora, using our approach and compared to the tree edit distance approach. In this comparison, only the results using the time model have been displayed, after checking that indeed, it outperformed the rest of approaches. This experiment has been performed under a query against database approach, so no deviations are provided. Note the clear improvement, especially for the *Essen-Lied* corpus.

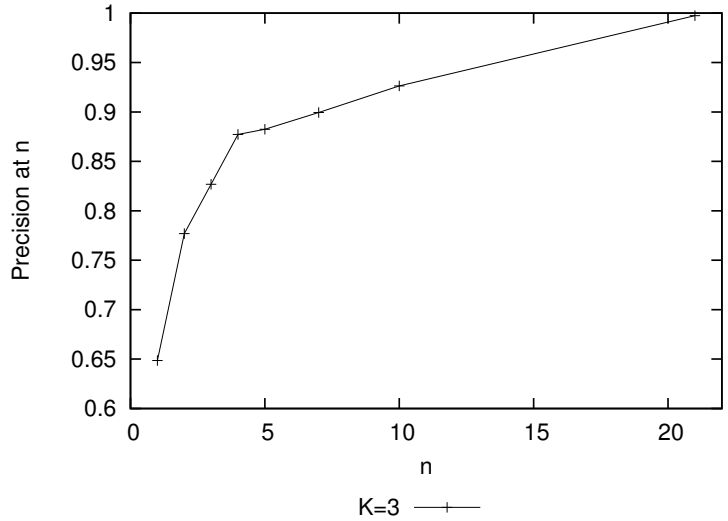


Figure 5.4: Success rates for $K = 3$ model as a function of the number of training samples (*Pascal*).

Table 5.4: Success rates (%) using the time model approach and comparison with tree edit distance for *Essen-Kinder* and *Essen-Lied* corpus.

	Tree edit dist.	$K = 2$	$K = 3$	$K = 4$
Essen-Kinder	90.0	89.9	97.3	99.7
Essen-Lied	59.0	96.3	100	100

5.5 Conclusions

In this work, we applied stochastic k-testable tree-models showing that they are suitable for the classification of tree-represented music data. Our goal was to identify a melody from a set of different variations. The results overcame those previously obtained using tree edit distances for the same corpora.

Based on the results we can say that the classification is improved when taking into account all the bars of the melody rather than using them separately. We can therefore say that stochastic k-testable tree models somehow capture the structure of the melody, so that this structure could be used to improve the melody classification.

Another important point here is efficiency and the scalability of the approach. It is remarkable that the classical tree edit distance has a high complexity, $O(m_A m_B \max\{h_A, h_B\})$, where m_i are the maximum arities

of the two trees, and h_i their heights, which in practice is actually $O(|M_A||M_B| \max\{h_A, h_B\})$. This must be multiplied by the complexity (quadratic) of the nearest neighbour method implemented to perform the classification, so if $n = |\Sigma|$ the total complexity is $O(n^2|M_A||M_B| \max\{h_A, h_B\})$. On the other hand, the proposed method has been shown (García, 1993) to run in $O(|M|^{k-1} n \log n)$ time and the classification is performed in linear time with $|t|$, once k is fixed.

We are persuaded that the promising results obtained can be improved by adjusting some parameters not studied yet, like using different and more sophisticated discount methods. In addition, stochastic k-testable tree-models can be updated with new samples and do not require too many training samples to get an acceptable success rate as seen in the results.

Also other music categorization problems like genre or style classification, and other challenges like representing long-term dependencies in melodies will be explored in the future.

6

Tree grammars

In this chapter we propose a solution when we have melodies represented by trees for the training but the duration information is not available for the input data. For that, we infer a probabilistic context-free grammar using the information in the trees (duration and pitch) and classify new melodies represented as a string of pitches. We aim to identify a snippet query of pitches among a set of songs stored in symbolic format.

6.1 Introduction

Humans are very good at recognizing previously known patterns, even if our perceived inputs are distorted, they are presented just partially, or in the presence of noisy data. This is often the situation in music comparison. Two issues are concerned to this problem: the similarity computation and the representation structure.

In Chapter 5 we used grammatical inference techniques to learn a tree language. Then, the probability of a tree (melody) to belong to a language (melody class) can be computed and used to classify it, taking a maximum likelihood decision among the languages (classes) learnt. To learn a language from trees, probabilistic k-testable tree models (Knuutila, 1993) were used. These models are a generalization of the k-gram models commonly used for strings. They have been used in a number of applications, like structured data compression (Rico-Juan et al., 2005) or information extraction from structured documents (like HTML or XML, for example) (Kosalaa et al., 2006).

However, probabilistic k-testable tree models have a limitation. They only can accept tree data as input and therefore they do not work in situations where a tree can not be built. This is the case when, even if we have trees in the database for training, queries are just a sequence of pitches, so the duration information is not available.

In this chapter, a solution to this problem is proposed. We infer a probabilistic context-free grammar using the trees (duration and pitch) in the training set and classify a query melody represented by pitch strings by using probabilistic context-free grammars (tree grammars (Verdu-Mas et al., 2005)) obtained from the probabilistic k-testable tree models (see Section 5). This way, the tree structure representing rhythm is captured by the grammar in the training phase and used in the string parsing.

Symbolic music retrieval from queries has been extensively studied in the MIR literature. In order to search a query in a dataset of songs, we can apply any of the well-known pattern matching algorithms, like local string editing to each of the songs in the dataset, and retrieve a ranking of most similar items. The main problem here is the efficiency when using large datasets, but with the advantage that it can find a partial or approximate occurrence of the query in any part of the dataset. On the other hand, a previous indexation of the dataset, e.g by means of motive extraction, solves the scalability issue, but motive extraction usually needs to work with exact repetitions, making very inaccurate to build robust indices from different interpretations of the same song. In addition, when searching only in motives, the music not present in the motives is hidden.

Our proposal is able to solve intrinsically these problems. The probabilistic grammar structure itself encodes both motives and melody variations, giving more weight to the most repeated themes, without the need to be exact, and enabling the possibility to learn from different renderings of the same song, leaving aside the need to encode motives. Besides, there is no difference between looking for a whole song or searching a small query in the dataset.

6.2 Probabilistic Context-Free Grammars

Probabilistic Context-Free Grammars (PCFGs) are statistical models based on context-free grammars, which are a well known type of formal grammars in formal language theory. The main attractive feature of PCFGs is that they describe probability distributions on strings and more importantly on tree structures of the same strings. This makes them suitable for structured prediction in a variety of areas where tree-like structures on strings have some meaning.

In this section, we formally define CFGs (Section 6.2.1) and PCFGs (Section 6.2.2). We also describe the used algorithms (Section 6.2.3) related to PCFGs that allow to:

- Checking if a particular string is generated by a given PCFG

- Computing the probability assigned by a PCFG to a string
- Finding the most probable parse tree for a string
- Computing the prefix probability of a string

6.2.1 Definitions

Defining Context-Free Grammars

A context-free grammar (CFG) is a tuple $\langle N, \Sigma, R, I \rangle$, where N is a set of non-terminal symbols, Σ is a set of terminal symbols (where $N \cap \Sigma = \emptyset$), $R \subseteq N \times (N \cup \Sigma)^*$ is a set of production rules and $I \subseteq N$ is a set of starting non-terminals. Unless otherwise specified, we use a, b, c, d for symbols in Σ, other lower-case Roman letters for strings in Σ^*, capital Roman letters for non-terminals (apart from G which will normally be used to denote a CFG), S for any starting non-terminal and Greek letters for strings in $(N \cup \Sigma)^*$. A production rule (A, α) of a CFG is written as $A \rightarrow \alpha$, where A is the left-hand side (LHS) of the rule and α is the right-hand side (RHS). Multiple production rules with the same RHS: $A \rightarrow \alpha_1, A \rightarrow \alpha_2, \cdots, A \rightarrow \alpha_n$, can be written as follows: $A \rightarrow \alpha_1 | \alpha_2 | \cdots | \alpha_n$. Usually, only the production rules of a CFG are written and the non-terminals, terminals and starting non-terminals are induced from the rules themselves according to the notation defined above. For example, the CFG $\langle N, \Sigma, R, I \rangle$ where $N = \{S_1, S_2, C, D\}, \Sigma = \{a, b, c, d\}, R = \{(S_1, aS_1b), (S_1, ab), (S_2, CD), (C, cC), (C, c), (D, dD), (D, d)\}$ and $I = \{S_1, S_2\}$ is simply written as follows:

- $S_1 \rightarrow aS_1b \mid ab$
- $S_2 \rightarrow CD$
- $C \rightarrow cC \mid c$
- $D \rightarrow dD \mid d$

The derivational relation over a CFG G, denoted as $\Rightarrow_G$, is a relation on $(\Sigma \cup N)^*$ defined as follows:

$$\alpha \Rightarrow_G \beta \; if \; \exists A \in N, \exists \gamma, \mu, \omega \in (\Sigma \cup N)^* : \\ (\alpha = \mu A \omega) \; and \; (\beta = \mu \gamma \omega) \; and \; (A \rightarrow \gamma \in R)$$

The transitive closure of $\Rightarrow_G$ is denoted as $\Rightarrow_G^+$ and the reflexive transitive closure of $\Rightarrow_G$ is denoted as $\Rightarrow_G^*$. The subscript is omitted whenever the

grammar G is clear from the context. A string $\alpha \in (N \cup \Sigma)^*$ derives another string $\beta \in (N \cup \Sigma)^*$ if $\alpha \Rightarrow_G^* \beta$. A string $\alpha \in (N \cup \Sigma)^*$ reaches another string $\beta \in (N \cup \Sigma)^*$ if for G any $\mu, \omega \in (N \cup \Sigma)^*$, $\alpha \Rightarrow_G^* \mu\beta\omega$. A grammar G derives (resp. reaches) a string $\alpha \in (N \cup \Sigma)^*$ if for any $S \in I$, S derives (resp. reaches) α.

The language generated from a non-terminal A is $L(A) = \{w \in \Sigma^* | A \Rightarrow^* w\}$ and the sentential forms generated from a non-terminal A are $\hat{L}(A) = \{\alpha \in (\Sigma \cup N)^* | A \Rightarrow^* \alpha\}$. The language generated by a CFG G is $L(G) = \cup_{S \in I} L(S)$ and the sentential forms of a grammar G are $\hat{L}(G) = \cup_{S \in I} \hat{L}(S)$. A context-free language (CFL) is any language generated by a CFG. Two CFGs are equivalent if they generate the same language.

A derivation of a non-terminal A, denoted D_A, is a sequence of length n of elements in $(N \cup \Sigma^*)$ such that $D_A[1] = A$, $D_A[n] \in \Sigma^*$ and for every $i \in 1 \cdots n-1$, $D_A[i] \Rightarrow_G D_A[i+1]$. A derivation of a grammar G is any D_S for $S \in I$. The subscript of D is omitted whenever we refer to a derivation in general (from any non-terminal). A leftmost derivation is a derivation D of length n such that for every $i \in 1 \cdots n-1$, the leftmost non-terminal of $D[i]$ is the one substituted with the RHS of a production rule to obtain $D[i+1]$. The set of leftmost derivations of a string $w \in \Sigma^*$ from a grammar G (resp. non-terminal A), denoted $LD_G(w)$ (resp. $LD_A(w)$), is made up of all the leftmost derivations of G (resp. A) of length n where $D[n] = w$. Every leftmost derivation can be graphically represented as a parse tree. For example, in Fig. 6.1 the parse tree for the leftmost derivation $S \Rightarrow SS \Rightarrow aSbS \Rightarrow aabbS \Rightarrow aabbab$ is showed:

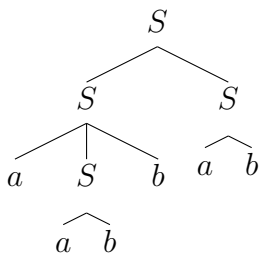


Figure 6.1: Parse tree for the leftmost derivation $S \Rightarrow SS \Rightarrow aSbS \Rightarrow aabbS \Rightarrow aabbab$.

For any derivation D of length n, $r(D)$ is a sequence of length $n-1$ of the production rules used in the derivation, where $r(D)[i]$ is the rule used to reduce $D[i]$ to $D[i+1]$. So, for the derivation $D = S \Rightarrow SS \Rightarrow aSbS \Rightarrow aabbS \Rightarrow aabbab$, $r(D)$ is $[S \to SS, S \to aSb, S \to ab, S \to ab]$.

The degree of ambiguity of a string $w \in \Sigma^*$ with respect to a grammar G is the number of leftmost derivations of w from G. A CFG G is said to be

ambiguous if at least one string has a degree of ambiguity bigger than 1. A CFL L is said to be inherently ambiguous if no unambiguous CFG is capable of generating L.

A context-free grammar is proper if it satisfies the following three conditions:

1. It is cycle-free, i.e. no non-terminal A exists such that $A \Rightarrow^+ A$.
2. It is λ-free, i.e. either no rules with λ on the RHS exist or exactly one exists with S on the LHS (i.e. $S \to \lambda$) and S does not appear on the RHS of any other rule.
3. It contains no useless symbols or non-terminals. This means that every terminal and non-terminal should be reachable from S and every non-terminal should derive at least one string from Σ^*.

6.2.2 Defining Probabilistic Context-Free Grammars

A probabilistic context-free grammar (PCFG) is a CFG with a function assigning a probability value to each production rule and to each starting non-terminal. Formally, a PCFG is a tuple $\langle G, P \rangle$ where G is the underlying CFG $\langle N, \Sigma, R, I \rangle$ and P is a function from $(R \cup I)$ to $[0, 1]$ s.t.

$$\forall A \in N, \sum_{A \to \alpha \in R} P(A \to \alpha) = 1$$

and

$$\sum_{S \in I} P(S) = 1$$

The probability of a derivation of a non-terminal A, denoted as $Pr(D_A)$, is the product of the probabilities of the rules used in the derivation:

$$Pr(D_A) = \prod_{A \to \alpha \in r(D_A)} P(A \to \alpha)$$

The probability of a derivation of a PCFG, denoted as $Pr(D_S)$, is the product of the probabilities of the rules used in the derivation and the starting non-terminal probability:

$$Pr(D_S) = P(S) \prod_{A \to \alpha \in r(D_S)} P(A \to \alpha)$$

The probability that a non-terminal A generates a string w, denoted as $Pr(A \Rightarrow^* w)$ and known as the inside probability of w from A, is the summation of the probabilities of all leftmost derivations of w from A:

$$Pr(A \Rightarrow^* w) = \sum_{D_A \in LD_A(w)} Pr(D_A)$$

The probability assigned by a PCFG G to a string $w \in \Sigma^*$, denoted $Pr(w)$, is the summation of the probabilities of all leftmost derivations of w from G:

$$Pr(w) = \sum_{D_S \in LD_G(w)} Pr(D_S)$$

6.2.3 Parsing and Probability of a String

An algorithm for finding the probability that a PCFG assigns to a string is crucial to have. Without this algorithm, we do not have access to the distribution defined by the PCFG over the language generated by the underlying CFG. Without access to the distribution, a PCFG is practically no better than its underlying CFG.

In Section 6.2.2, we defined the probability assigned by a PCFG to a string as the summation of the probabilities of all leftmost derivations. This definition does not directly translate into an efficient procedure. First of all, it does not describe how derivations can be found and secondly, the number of leftmost derivations can be exponential (or even infinite) in the length of the string.

Fortunately, efficient procedures (that use dynamic programming) exist to solve this problem. These procedures were initially designed to find and efficiently store all the leftmost derivations of a given string w from a given grammar G; in other words, they were designed as parsers for CFGs. These procedures can be easily extended to work with PCFGs by also returning the probability of the given string. In this section, we describe and analyse the procedure used and show how they can be used to find the probability of a string.

There are mainly two procedures to solve this problem, the CockeY-oungerKasami (CYK) algorithm and the Earley algorithm. On the one hand, the CYK algorithm assumes that the grammar to be parsed is in Chomsky Normal Form (CNF). A grammar is in CNF if every production rule is in $(N \times N^2)$ or $(N \times \Sigma)$. For every CFG G, there exists an equivalent CFG G' in CNF (Harrison, 1978). An equivalent CFG in CNF can be built in polynomial

time from any CFG following the steps described in (Rich, 2008). Attending to the complexity, the CYK algorithm and its extensions take $O(|P|n^3)$ time, where $|P|$ is the number of production rules of the grammar to be parsed and n is the length of the string to be parsed. The n^3 term stands for the fact that every split of every substring is traversed, whilst the $|P|$ term is there because all production rules are traversed to find matching rules for each split. The space complexity is $O(n^2 \cdot |N|)$, which is the size of the CYK table. On the other hand, the Earley algorithm, unlike the CYK algorithm, it does not assume that the grammar to be parsed is in CNF. Moreover, the worst case running time complexity of the Earley algorithm is $O(|G|^2 n^3)$ (An improved Earley algorithm is given in (Graham et al., 1980) with a worse case running time of $O(|G|n^3)$). However, for certain types of grammars, the Earley algorithm has a lower worst time complexity. For example, the complexity is $O(|G|^2 n^2)$ for grammars with bounded ambiguity and $O(|G|^2 n)$ for deterministic grammars (Stolcke, 1995). The space complexity in terms of the length of the string is $O(n^2)$.

Taking into account the above, in this work the Stolcke algorithm, a probabilistic extension of the Earley algorithm, is used. In this way, we avoid the use of an intermediate procedure to transform a PCFG in Chomsky Normal Form and solve different probabilities in a common framework as we show in the next section.

Earley (Stolcke) Algorithm

Unlike the CYK algorithm, the Earley (Stolcke) algorithm is considered as a top-down rather than a bottom-up parsing algorithm. This is because it works its way down from the starting production rules to the symbols in the string rather than the other way round.

Although the Earley algorithm can be modified to work with unrestricted CFGs, the standard Earley algorithm works with particular forms of grammars used to model natural language syntax. These grammars satisfy the following constraints:

- Every production rule is of the form $N \rightarrow N^+$ or $N \rightarrow \Sigma$
- The set of non-terminals can be split into two disjoint sets, parts of speech (POS) non-terminals and phrasal non-terminals. The LHS non-terminals of rules of the form $N \rightarrow \Sigma$ are POS non-terminals while the rest are phrasal non-terminals.

Note that any CFG (resp. PCFG) can be transformed into an equivalent CFG (resp. PCFG) which satisfies these constraints. This can easily be done

by substituting terminals a in RHS of rules of length bigger than 1 with newly added non-terminal N_a and adding POS rules $N_a \to a$. Probabilities can easily be assigned in such a way that the resulting grammar remains probabilistically equivalent.

The Earley algorithm works by traversing the given string $w = a_0 \dots a_{n-1}$ once from left to right and adding states to a chart as it goes along. States show how much a particular production rule has been processed so far and which substring it generates at that point. The presence of some particular states in the last chart entry indicates that the string is recognized by the grammar. We will first explain the syntax and semantics of a state, followed by an explanation of what the chart is, and finally we show how the chart is filled with states and which states determine whether a string is recognized or not by the grammar.

The states and the chart A state takes one of the following two forms:

$$[A \to \alpha \bullet B\beta, i, j]$$

$$[A \to a_i \bullet, i, i+1]$$

where:

- $A \to \alpha B \beta$ and $A \to a_i$ are production rules (where $\alpha, \beta \in N^*$)
- $A \to \alpha \bullet B\beta$ and $A \to a_i \bullet$ are known as dotted rules. The $\bullet$ symbol for the first rule can also be at the end of the rule (i.e. $A \to \alpha \bullet$, where $\alpha \in N^+$)
- i and j are two indices from $[0, n-1]$ and $[0, n]$ respectively such that $i \leq j$

A state $[A \to \alpha \bullet B\beta, i, j]$ indicates two things:

1. Production rule $A \to \alpha B \beta$ has been processed up until the last non-terminal of α. If α is empty, then the production rule has not yet been processed and if $\bullet$ is at the end of the rule then the production rule has been fully processed.
2. $\alpha \Rightarrow^* a_i \dots a_{j1}$. In case α is empty, then the production rule does not generate anything (since it has not yet been processed) and thus i is equal to j.

The chart C is an array with $n + 1$ entries (from 0 to n). Each entry holds an ordered set of states. $C[j]$ holds states with index j (i.e. states whose production rules generate substrings up to a_{j1}). $C[0]$ contains states with unprocessed production rules starting from the beginning of the string and $C[n]$ contains fully processed production rules generating suffixes of w. Having a state $[S \rightarrow \alpha\bullet, 0, n]$ in $C[n]$ (for any $S \in I$) means that the grammar generates w.

Filling the chart The chart is incrementally filled in n steps, starting from $C[0]$ up to $C[n]$. A state with the second index j will be in $C[j]$. States are never removed from the chart. Three operators are used to fill the chart: the PREDICTOR, SCANNER and COMPLETER. Each operator takes one state as input and constructs new states from it. The COMPLETER needs access to states in previous chart entries, whilst the PREDICTOR and SCANNER do not need this. At the j^{th} step, the PREDICTOR and COMPLETER add states to $C[j]$ whilst the SCANNER adds states to $C[j + 1]$. The three operators work as follows:

- Given a state $[A \rightarrow \alpha \bullet B\beta, i, j]$ where B is a phrasal non-terminal, the PREDICTOR adds the states $[B \rightarrow \bullet\gamma, j, j]$ in $C[j]$ for every production rule $B \rightarrow \gamma$ with B on the LHS. Therefore, the PREDICTOR adds states with all the possible rules which the non-terminal just after the $\bullet$ can use. Note that the rules in the added states are unprocessed since the indices are the same.

- Given a state $[A \rightarrow \alpha \bullet B\beta, i, j]$ where B is a POS non-terminal, the SCANNER adds the state $[B \rightarrow a_j\bullet, j, j + 1]$ in $C[j + 1]$ if there exists a production rule $B \rightarrow a_j$.

- Given a state $[B \rightarrow \alpha\bullet, k, j]$ (i.e. a state with a fully processed production rule), the COMPLETER searches for states in $C[k]$ of the form

  $$[A \rightarrow \alpha \bullet B\beta, i, k]$$

  and for each such state, the state:

  $$[A \rightarrow \alpha B \bullet \beta, i, j]$$

  is added to $C[j]$. In other words, the COMPLETER adds states with the $\bullet$ symbol moved one step to the right (i.e. from behind B to after B) when it finds a fully processed rule for B with matching indices.

Algorithm 1 shows how and when these three operators are used to fill the whole chart. Note that the initial states in $C[0]$ have unprocessed rules with starting non-terminals on the LHS. Also note that each time a state is added in a chart entry, it is placed at the end of the ordered set. Therefore, when iterating on the same set where states are added, the newly added states will always be reached by the iteration. After filling the chart, we know that w is accepted by the grammar if there exists a state $[S \to \alpha\bullet, 0, n]$ in $C[n]$ (for any $S \in I$), otherwise w is not accepted by the grammar.

The probability of the string can be calculated by multiplying the probabilities associated with successive states in the Earley chart. In addition to providing a solution to calculate the probability that a given string x is generated by a grammar G, the Stolcke algorithm can compute solutions to know what is the single most likely parse (or derivation) for x or to get the probability that x occurs as a prefix of some strings generated by G (the prefix probability of x). All of these solutions in a single framework, with a number of additional advantages over isolated solutions (see (Stolcke, 1995) for more details).

6.3 Stochastic k-testable Tree Grammars

As shown in Chapter 5, a probabilistic DTA (PDTA) incorporates a probability, $p_m(\sigma, t_1, ..., t_m)$, for every transition in the automaton, with the normalization that the probabilities of the transitions leading to the same state $q \in Q$ must add up to one. These probabilities must be calculated as the ratio between the number of occurrences of a transition and the number of occurrences of the state to which this transition leads. PDTA also incorporates a probability $\rho(q)$ for every accepting state, $q \in F$, calculated as the ratio between the number of occurrences of an accepting state and the number of trees in the sample, $|\Omega|$.

Eventually, the probability of the tree t is computed as $p(t) = \rho(\delta(t))\pi(t)$, where the product of the probabilities of all the transitions is recursively computed as:

$$\pi(t) = \begin{cases} p_0(\sigma) & \text{if } t = \sigma \in \Sigma \ , \\ p_m(\sigma, \delta(t_1), ..., \delta(t_m))\pi(t_1) \cdots \pi(t_m) & \text{if } t = \sigma(t_1...t_m) \in T_\Sigma - \Sigma \ . \end{cases} \tag{6.1}$$

At this point, we can classify a new melody in a particular class. For this purpose, we need to infer a PDTA for each class, C_j, from well classified melodies. Once the PDTAs for the different classes have been inferred and the probabilities estimated, a melody M can be classified in the class $\hat{C}$ that maximizes the likelihood after parsing with every PDTAs.

Algorithm 1: Earley Algorithm

Input : A CFG G; a string $w = a_0 \dots a_{n-1}$
Output: True if $w \in L(G)$, False otherwise
$C[0] \leftarrow \{[S \rightarrow \bullet\alpha, 0, 0] | S \in I, S \rightarrow \alpha\}$;
for $i = 0$ *to* n **do**
  foreach *state in* $C[i]$ **do**
    if *production rule in state is fully processed* **then**
      COMPLETER(state);
    else
      if *the non-terminal after* $\bullet$ *is a POS non-terminal* **then**
        SCANNER(state ;
      else
        PREDICTOR(state) ;
if $\{[S \rightarrow \alpha\bullet, 0, n] | S \in I, S \rightarrow \alpha\} \cap C[n] = \emptyset$ **then**
  return False ;
else
  return True ;
procedure *COMPLETER(*$[B \rightarrow \alpha\bullet, k, j]$*)*
  foreach *state in* $C[i]$ **do**
    Add $[A \rightarrow \alpha B \bullet \beta, i, j]$ to $C[j]$;
procedure *SCANNER(*$[A \rightarrow \alpha \bullet B\beta, i, j]$*)*
  if $B \rightarrow a_j \in R$ **then**
    Add $[B \rightarrow a_j\bullet, j, j+1]$ to $C[j+1]$;
procedure *PREDICTOR(*$[A \rightarrow \alpha \bullet B\beta, i, j]$*)*
  foreach $B \rightarrow \gamma \in R$ **do**
    Add $[B \rightarrow \bullet\gamma, j, j]$ to $C[j]$;

In order to do this, the new target melodies must be represented by trees for being parsed. However, what happens if the new melodies can not be represented by trees and are only strings? This situation appears when a melody query is given using only note pitches or when durations are not available or reliable. In other words, we have a set of melodies represented by trees to train the system but the target data are melodies represented by pitch strings. To deal with this problem, we need to transform the k-testable tree automata in context-free grammars in order to use them for parsing the input melody strings.

Context-free grammars may be considered to be the customary way of representing syntactical structure in natural language sentences. In many natural language processing applications, to obtain the correct syntactical structure for a sentence is an important intermediate step before a semantic interpretation. Therefore, we need to transform the k-testable tree automata in probabilistic context-free grammars to use them for parsing the input melody strings. Thus, we could use these grammars to obtain the correct structure for a given melody represented by a pitch string.

These context-free grammars obtained from the k-testables tree models include some degree of specialization in its rules that weaken it independence. In these way, they are specially attractive, on one hand, the relations in long term between the language components are captured by its grammar condition. On the other hand, the local relations are represented by its k-testable condition. Moreover, this last fact rekindle in a structural ambiguity reduction.

Before transforming our k-testable tree automata in context-free grammars we need introduce some changes in the melody tree representation. These changes are necessary because if we use the alphabet described in Section 3.3 then a transition in the automaton could be transformed in different grammar rules. This happens because we can not distinguish between symbols that are terminals or nonterminals. In order to solve these ambiguities we need to label tree nodes adding the symbol 'T' to that of Σ_p if it is a leaf node (terminal) and the symbol 'N' if it is an inner node (nonterminal). This change also solve the notation requirements to use the Stolcke algorithm where the LHS non-terminals of rules of the form $T \to \Sigma$ are POS non-terminals while the rest are phrasal non-terminals as we showed in Section 6.2.3.

Therefore, if $\Omega = \{t_1, t_2, \ldots, t_{|\Omega|}\}$ is a treebank, that is, a stochastic sample of parse trees, the alphabet Σ can be safely partitioned into the subset $s_1(\hat{\Omega})$ of labels that may only appear at leaves ($\Sigma_{pT} = \text{'}T\text{'}\Sigma_p$) and its

complementary subset $\Sigma - s_1(\hat{\Omega})$ (Σ_{pN} ='N'Σ_p) of labels at internal nodes (propagated labels in the trees).

Then, we can define a probabilistic k-testable grammar as $G^{[k]} = (V^{[k]}, T, I, R^{[k]}, p^{[k]})$, where I is the start symbol, $V^{[k]} = I \cup r_{k-1}(f_k(\Omega) \cup s_{k-1}(\hat{\Omega})) - s_1(\hat{\Omega})$ is the set of nonterminals, $T = s_1(\hat{\Omega})$ is the set of terminals, $R^{[k]}$ is the set of production rules, and $p^{[k]}$ a probability function, built as follows:

- For every tree $t \in r_{k-1}(\hat{\Omega})$, add the rule $I \to t$ to $R^{[k]}$ and compute its probability as

$$p^{[k]}(i \to t) = \frac{1}{|\Omega|} \Sigma_{n=1}^{|\Omega|} D(t, r_{k-1}(\tau_n))$$

  where $D(i, j) = 1$ if $i = j$ and zero otherwise.

- For every tree $\sigma(t_1, t_2 \cdots t_m) \in f_k(\hat{\Omega})$, add the rule $r_{k-1}(\sigma(t_1, t_2 \cdots t_m)) \to t_1 t_2 \cdots t_m$ to $R^{[k]}$ and compute its probability as

$$p[k](r_{k-1}(\sigma(t_1, t_2 \cdots t_m)) \to t_1 t_2 \cdots t_m) = \frac{\Sigma_n E^{[k]}(\sigma(t_1, t_2 \cdots t_m), \tau_n)}{\Sigma_n E^{[k-1]}(\sigma(t_1, t_2 \cdots t_m), \tau_n)}.$$

  where

$$E^{[k-1]}(j, \tau_n) = \sum_{i \in Q} \Phi(i, \tau_n) D(r_{k-1}(i), r_{k-1}(j))$$

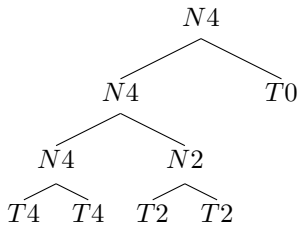
  and $\Phi(t, \tau)$ counts the number of nodes τ that expand a subtree s such that $s = \tau$.

- For every tree $\sigma(t_1, t_2 \cdots t_m) \in s_{k-1}(\hat{\Omega}) - T$, add the rule $\sigma(t_1, t_2 \cdots t_m)) \to t_1 t_2 \cdots t_m$ to $R^{[k]}$, its probability being one:

$$p[k](\sigma(t_1, t_2 \cdots t_m) \to t_1 t_2 \cdots t_m) = 1.$$

The above rules and probabilities are analogous to those given in 5.2.3 for the case of tree automata and define a consistent probabilistic context-free grammar (see (Verdu-Mas et al., 2005) for details). With this definition, when $k = 2$, only the label of the node is taken into account and the k-testable model coincides with the simple rule-counting approach used in treebank grammars (Charniak, 1996).

As an illustration, Figure 6.6 shows the corresponding probabilistic context-free grammars (PCFG) (right) for $k = 2$ and $k = 3$ derived from the parsing tree in the left. For this tree we have the following sets for the $K = 2$ model: (roots) $r_1(t) = N4$, (forks) $f_2(t) = \{N4(N4\ \ T0), N4(N4\ \ N2), N4(T4\ \ T4), N2(T2\ \ T2)\}$, (subtrees) $s_1(t) = \{T0, T2, T4\}$.



$$
\begin{array}{l}
(1/1)\ I \rightarrow N4 \\
(1/3)\ N4 \rightarrow N4\ \ T0 \\
(1/3)\ N4 \rightarrow N4\ \ N2 \\
(1/3)\ N4 \rightarrow T4\ \ T4 \\
(1/1)\ N2 \rightarrow T2\ \ T2
\end{array}
\Big\}\ G^{[2]}
$$

$$
\begin{array}{l}
(1/1)\ I \rightarrow N4(N4\ T0) \\
(1/1)\ N4(N4\ T0) \rightarrow N4(N4\ N2)\ \ T0 \\
(1/1)\ N4(N4\ N2) \rightarrow N4(T4\ T4)\ \ N2(T2\ T2) \\
(1/1)\ N4(T4\ T4) \rightarrow T4\ \ T4 \\
(1/1)\ N2(T2\ T2) \rightarrow T2\ \ T2
\end{array}
\Big\}\ G^{[3]}
$$

Figure 6.2: Example of a probabilistic tree grammar for $k = 2$, and $k = 3$.

6.3.1 Smoothing

In general, k-testable grammars with larger values of k contain more specialized rules and, therefore, are less ambiguous and allow for faster parsing. In contrast, smaller k provide more general grammars with more ambiguity and therefore parsing is slower, since all possible combinations need to be explored.

But the larger they are, the more likely they can find unseen strings, that will be assigned a zero probability in recognition. This would lead to the conclusion that a target melody is just impossible, but instead we have to say that it is less probable for that class. Therefore, the use of smoothing techniques becomes necessary if one wants to use these models for parsing.

Two classical smoothing techniques are linear interpolation and backing-off (Ney et al., 1995). Smoothing through linear interpolation is performed by computing the probability of events as a weighted average of the probabilities given by different model sizes. But if the higher k models return a zero probability, the target melody will be classified only with the more general and more ambiguous model discarding the entire, more specific, k-sized model.

In contrast, backing-off allows to avoid lower-order parsing when it is possible, because it tries to parse with the higher-order grammar unless no parse tree is available by this grammar. Only in such a case the lower-order model is called, so backing-off is faster than linear interpolation. However, the lack of a single rule in the sample can force the parser to use the lower-order model, loosing all the higher-order information for a whole melody.

In this work, we use an alternative approach: the rule-based backing-off (Verdu-Mas et al., 2005) similar to the approach used in 5.3.2 for the

k-testable tree automata. In this approach, the set of rules is generalized with the probabilities

$$p((r_{k-1}(\sigma(t_1, t_2 \cdots t_m)) \rightarrow t_1 t_2 \cdots t_m | G^{[k]}_{RULE}) = \begin{cases} (1-\lambda)p^{[k]}(r_{k-1}(\sigma(t_1, t_2 \cdots t_m)) \rightarrow t_1 t_2 \cdots t_m) \\ \quad if\ p^{[k]}(r_{k-1}(\sigma(t_1, t_2 \cdots t_m)) \rightarrow t_1 t_2 \cdots t_m) > 0 \\ \frac{\lambda}{\Lambda(r_{k-1}(\sigma(t_1, t_2 \cdots t_m)))} \prod_{i: r_{k-2}(t_i) \neq t_i} p^{[k-1]}(r_{k-2}(\tau_i) \rightarrow \tau_i) \quad otherwise, \end{cases} \tag{6.2}$$

where $t_1, \cdots, t_m$ are parse trees and

$$\Lambda(r_{k-1}(\sigma(t_1, t_2 \cdots t_m))) = 1 - \sum_{r_{k-1}(\sigma(t_1, t_2 \cdots t_m)) \rightarrow \tau_1 \cdots \tau_m \in R^{[k]}} \prod_{i: r_{k-2}(t_i) \neq t_i} p^{[k-1]}(r_{k-2}(\tau_i) \rightarrow \tau_i) \tag{6.3}$$

It requires the implementation of specific parsers, since building the whole grammar is unfeasible due to the large number of implicit rules. An alternative scheme that requires only minor modifications is to use a quasi-equivalent grammar G' built as follows:

1. Add the rules in $R^{[k]}$ to R' with probabilities

$$p'(X \rightarrow \alpha) = (1-\lambda)p^{[k]}(X \rightarrow \alpha).$$

2. For every variable $\sigma(t_1, t_2 \cdots t_m)$ in $V^{[k]}$, add a rule $\sigma(t_1, t_2 \cdots t_m)) \rightarrow t_1 t_2 \cdots t_m$ to R' whose probability is

$$p'(\sigma(t_1, t_2 \cdots t_m)) \rightarrow t_1 t_2 \cdots t_m) = \frac{\lambda}{\Lambda(\sigma(t_1 t_2 \cdots t_m))}$$

3. For every rule $r_{k-2}(\sigma(t_1, t_2 \cdots t_m)) \rightarrow t_1 t_2 \cdots t_m$ in $R^{[k-1]}$, add $r_{k-2}(\sigma(t_1, t_2 \cdots t_m)) \rightarrow r_{k-1}(\sigma(t_1, t_2 \cdots t_m))$ to R' and the probability

$$p'(r_{k-2}(\sigma(t_1, t_2 \cdots t_m)) \rightarrow r_{k-1}(\sigma(t_1, t_2 \cdots t_m)) = p^{[k-1]}(r_{k-2}(\sigma(t_1, t_2 \cdots t_m)) \rightarrow t_1 t_2 \cdots t_m)$$

As shown in Fig. 6.3, the parse trees generated by this grammar G' and their probabilities are, after an adequate projection ζ, identical to those generated by using the equation 6.2: If a node with label X has as single descendent a subtree $Y(\alpha)$ and the depth of label X is smaller than that of label Y, the projection operation ζ has to be applied so that $\zeta(X(Y(\alpha))) = Y(\alpha)$. Then, G' can be used with a standard chart parser provided that some care is taken to avoid selecting a projected subtree whenever the same subtree

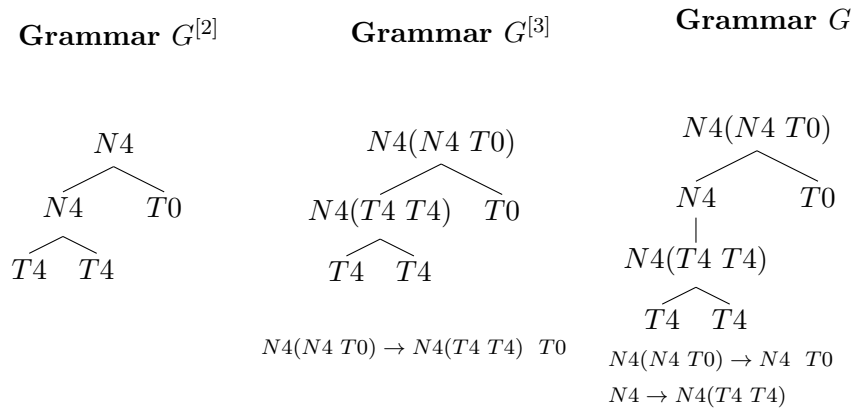


Figure 6.3: Parsing example

can be obtained without projection. This can be checked by redefining the suitable comparison operator so that t_1 is not a better tree than t_2 if $\zeta(t_1) = \zeta(t_2) = t_2$ (even if $p(t_1|G') > p(t_2|G')$.

In this procedure the different k-grammars are combined to build a unique grammar for each class C_j. In Fig. 6.4 a little schema is showed where the k-grammars inferred from the tree data (pitch and duration) with different values of k are merge in a unique grammar that include all the rules. This grammar is used to parse the strings (only pitch) in the test data.

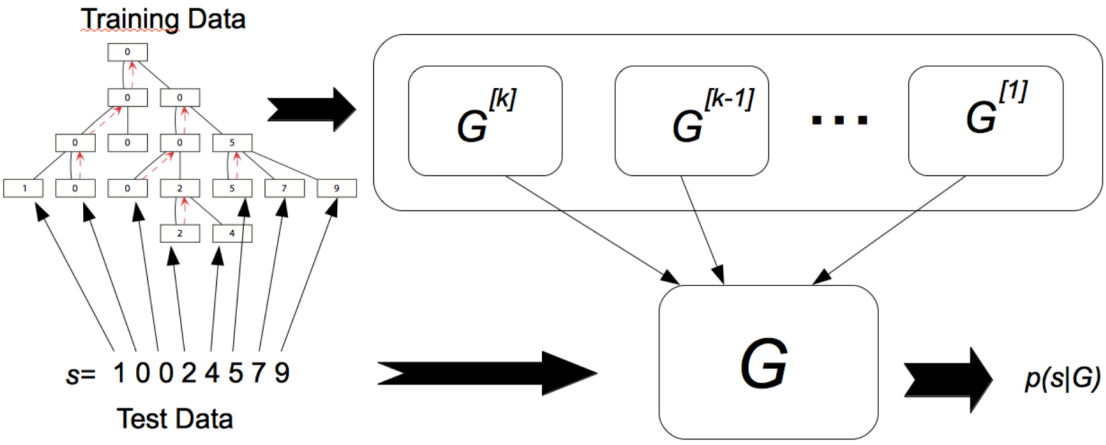


Figure 6.4: Schema of a unique combined grammar that merge the different k-grammars.

$G^{[3]}$
$I \rightarrow N4(N4\ T0)$
$N4(N4\ T0) \rightarrow N4(N4\ N2)\ \ T0$
$N4(N4\ N2) \rightarrow N4(T4\ T4)\ \ N2(T2\ T2)$
$N4(T4\ T4) \rightarrow T4\ \ T4$
$N2(T2\ T2) \rightarrow T2\ \ T2$

$G^{[2]}$
$N4 \rightarrow N4\ \ T0 \Rrightarrow N4 \rightarrow N4(N4\ T0)$
$N4 \rightarrow N4\ \ N2 \Rrightarrow N4 \rightarrow N4(N4\ N2)$
$N4 \rightarrow T4\ \ T4 \Rrightarrow N4 \rightarrow N4(T4\ T4)$
$N2 \rightarrow T2\ \ T2 \Rrightarrow N2 \rightarrow N2(T2\ T2)$

Figure 6.5: Example of how the $k-1$ rules are changed in a model with $K = 3$.

Each rule in the k model is added to the new grammar G' with its probability minus a portion, this portion being the discount parameter λ. Afterwards, the backing-off drops into the $k-1$, adding the needed rules with a probability determined by the discount parameter. That is, the probability mass that is removed to the k rules is added to the backing-off rules to drop into the $k-1$ model. This procedure is performed for each k. In Fig. 6.5 an example of how the $k-1$ rules are changed in a model with $K = 3$. Note the k rules (in this case $k = 3$) do not change while the $k-1$ are changed in the suitable form.

However, we need to define a universal grammar because some labels in Σ_p for the leaves of the trees do not appear in the training data for the grammars. Then, if a particular new melody contains an unseen label, the parser will return a zero probability. This particular problem is more relevant in music than in texts, because if training melodies are in major and minor modes (the most common in many genres), some degrees may not appear in the whole training set (like minor second, 1 in Σ_p, or diminished fifth, 6 in Σ_p, for example), but they can in a target query, and the models can not deal with such a case.

For solving this problem we introduce rules of the form $N\sigma_1 \rightarrow N$ (where N is a new symbol used as a backing-off rule for the model $k = 1$), where the probability is given by the frequency of the symbol $N\sigma_1$ in the training data. Then, the rules of the form $N \rightarrow T\sigma_2$ are added with a probability given by each symbol. Since the alphabet Σ_p is finite, each symbol $T\sigma_2$ is

$G^{[3]}$
$I \rightarrow N4(N4\ T0)$
$N4(N4\ T0) \rightarrow N4(N4\ N2)\ \ T0$
$N4(N4\ N2) \rightarrow N4(T4\ T4)\ \ N2(T2\ T2)$
$N4(T4\ T4) \rightarrow T4\ \ T4$
$N2(T2\ T2) \rightarrow T2\ \ T2$

$G^{[3]} \Longrightarrow G^{[2]}$
$N4(N4\ T0) \rightarrow N4\ \ T0$
$N4(N4\ N2) \rightarrow N4\ \ N2$

$G^{[2]}$
$N4 \rightarrow N4(N4\ T0)$
$N4 \rightarrow N4(N4\ N2)$
$N4 \rightarrow N4(T4\ T4)$
$(N2 \rightarrow N2(T2\ T2)$

$G^{[2]} \Longrightarrow G^{[1]}$
$N0 \rightarrow N$
...

$G^{[1]}$
$N \rightarrow T0$
$N \rightarrow N0$
...

Figure 6.6: Example of G' for $K = 3$.

initialized with its count to 1. This way, if a symbol $T\sigma_2$ does not appear in the training data, it will remain with a low probability but not zero, solving the problem of unseen labels. Note that these rules only work with terminals $T\sigma_2$, hence, the ambiguity is only allowed in the leaves of the tree, and the structure of the parsing trees is preserved.

Finally, in Fig. 6.6 a complete grammar G' for $K = 3$ is showed. In this example the rules of the models $k = 3, k = 2$ and $k = 1$ (universal grammar) (on the left) are combined with its respective backing-off rules showed on the right. Once this grammar is built for each class, we can proceed to classify new melodies represented as strings.

6.4 Classification

As explained before, we want to study if the proposed approach can be used to classify new melodies represented by strings of pitches. After a grammar G_j is inferred for each class C_j we need an algorithm for obtaining the probability that a given string s is generated by G_j. For this purpose, we have used the

Stolcke algorithm (Stolcke, 1995) for string parsing described in Section 6.2.3. This parsing algorithm are able to give the probability $p(s|G)$ that a string s is generated by a probabilistic Context-free grammar G without requiring conversions to Chomsky Normal Form (CNF). Then, the string representing the melody M is classified in the class $\hat{C}$ that maximizes the likelihood

$$\hat{C} = \arg\max_j \ l(M|C_j) \tag{6.4}$$

We can calculate this likelihood in two ways: splitting the melody (SplitBars) in bars or computing the whole melody (Whole).

In SplitBars, the melody string is split in $|M|$ (number of bars in a melody M) bar strings $s_1, \ldots, s_{|M|}$. Therefore we are able to compute the probability of each bar string to belong to a particular class (grammar). Suppose we have a finite number of classes and we have computed the membership probability of each bar string s_i to each of these grammars, $p(s_i|G_j)$. These probabilities can be combined to give a decision for the whole song. For the combination of bars the geometric mean of the probabilities has been used. The geometric mean is less sensitive to outliers than the arithmetic one. For our purposes is enough to multiply all bar strings probabilities of the whole melody. Calculating the $|M|$-th root of the resulting product, required by the geometric mean, is not needed for classification because, given a particular melody M to classify, $|M|$ is the same for all classes. Therefore, the likelihood is computed as

$$l(M|C_j) = \prod_{i=1}^{|M|} p(s_i|G_j) \tag{6.5}$$

On the other hand, in the 'Whole' strategy we only need the probability of the melody string (now $M = s$). Then

$$l(M|C_j) = p(s|G_j) \tag{6.6}$$

For calculating this probability we need to introduce the start rules $S \to IS$ and $S \to I$ which define a melody recursively (melody is formed by a bar and a melody) (I is the initial symbol for a bar, as explained in Section 6.3). Therefore, the grammars allow to recognize a whole melody instead of melodies split in bars.

The solutions presented above have some drawbacks. In the 'SplitBars' strategy the system needs to know where the bars begin. But this information usually is not available in a query made of pitch symbols and an algorithm to partition the string melody is needed. However we do not know any method able to do that only with a string of pitches, without the duration and meter

information. On the other hand, the 'Whole' strategy has problems when the melody is very large, due to the time complexity of the parsing algorithms. In the case of the Stolcke algorithm the time complexity is $O(|M|^3)$.

However, the Stolcke algorithm computes the probability that a given string s is generated by a grammar G and the probability that x occurs as a prefix of some string generated by G (the prefix probability of x). Computations for these tasks proceed incrementally, as the parser scans its input from left to right. In particular, prefix probabilities are available as soon as the prefix has been seen, and are updated incrementally as it is extended. When the string is completely parsed both probabilities are the same. This way, we can use Stolcke to calculate, in parallel, the prefix parse for each class and make a ranking of classes for a given melody. The ranking allows the system to stop the parsing for the worst classes when a given number of melody symbols have been parsed. In other words, if we have a long melody, we could classify it using only the beginning of the query. After that, we compute the probability of the whole melody only for the most probable classes. This way, we can improve the computing time for long melodies.

6.5 Results

In our experiments, we have tried to identify a target melody using a set of different variations played by musicians for training the models for the different classes. We have tested our methodology using three different corpora.

The first corpus, named *Pascal* database consisting of a set of 420 monophonic 8-12 bar incipits of 20 worldwide well known tunes of different musical genres (see Section 3.4.1 for details).

Two other corpora (*Essen-Kinder* and *Essen-Lied*) have been used for testing the proposed method in order to test its performance on other kind of variations automatically generated (see Section 3.4.2 for details).

A 3-fold cross-validation scheme was carried out to perform the experiments, obtaining average success rates and dispersions $((max-min)/4)$. The system learn a grammar G' with $K = 3$ ($G^{[3]}, G^{[2]}, G^{[1]}$, and the corresponding backing-off rules) with a discount $\lambda = 0.1$.

Table 6.1 shows the results of classification using the approaches explain in Section 6.4. These results are compared with the results using the approach of probabilistic deterministic tree automata used in Chapter 5 but using the notation change described in Section 6.3.

Table 6.1: Success rates with the different approaches used.

Approach	Success rate
PDTA	87.3 ± 0.7
StringBars	92.4 ± 1.1
Whole	86.9 ± 1.3

Note that PDTA uses the duration information implicit in tree representation, however the grammar approaches use less information (only pitch) for classifying. From the results, it is observed that the new approach using the strings through the StringsBars method improves significantly the PDTA results. The Whole approach did not improve the results because it is more sensitive to variations in the data than the geometric mean of the bar probabilities.

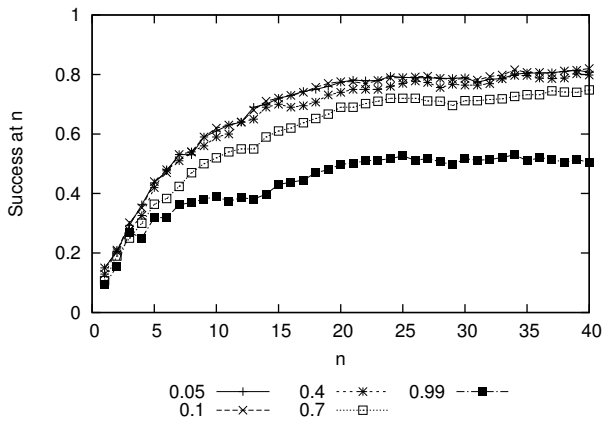


Figure 6.7: Success rate behaviour in function of the number of symbols processed for several values for the discount parameter λ.

Figure 6.7 shows the success rate behaviour as a function of the number of symbols processed for several values of the discount parameter λ. Here we can see the system ability to classify queries of a given length $n = |M|$, instead of the whole melody. This way, the system can obtain a useful success rate without computing the probability of the whole melody. Thus, computing the prefix probability of n symbols is enough. A good result is, for example, the obtained for a prefix of 18 symbols with a success rate of 0.757 ± 0.011. This way, the system avoids the whole melody computation. Average $|M|$ in this corpus, is about 100 pitch symbols.

About the discount parameter, we can see that its behaviour is as expected. That is, when the discount parameter is high the success rate decreases. This is because if the system takes out more probability mass of the more specific models to add this probability to the more general models, the system becomes an ambiguous classifier without discrimination power giving more weight to the rules of the universal ($k = 1$) model instead the more specific models.

Figure 6.8 shows the success rate when the proper class is in the best i classes ($i = \{1..4\}$). The system returns a ranking sorted by the prefix probability parsed at each time. This ranking can be used in two ways. On the one hand, to retrieve the i best classes if is required from a particular task obtaining a success rate of about 95% for a prefix of 18 symbols. On the other hand, the ranking can be used to stop the computation of the worst classes when a given number of melody symbols have been parsed while the system continues computing the probability in the remain classes. After that, we compute the probability of the whole melody only for the best classes, improving the computation time.

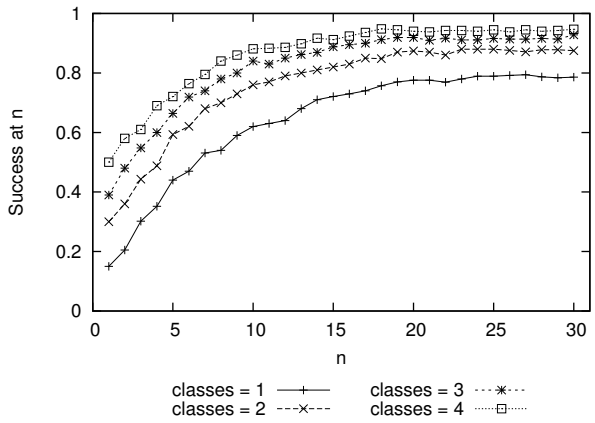


Figure 6.8: Success rate when the proper class is in the best i classes.

Figures 6.9 and 6.10 show the success rate for the *Essen-Lied* and *Essen-Kinder* corpora, using our approach. This experiment has been performed under a query against database approach, so no deviations are available. Note the clear improvement, especially for the *Essen-Kinder* corpus.

6.6 Conclusions

In this work, we applied probabilistic tree grammars constructed from stochastic k-testable tree-models showing that this approach can be used

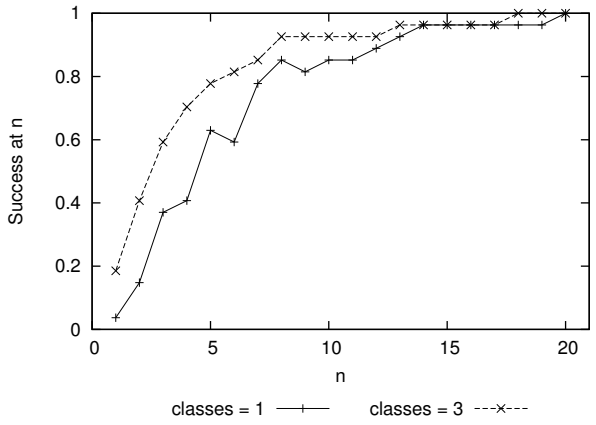


Figure 6.9: Success rate when the proper class is in the best i classes (*Essen-Lied*) .

for classifying new melodies represented by strings using the information captured in the grammar rules. This approach allows avoiding the duration information in the input data (strings with pitch only), making easier querying a music database. Our goal was to identify a melody from a set of different variations. The results overcame those previously obtained using probabilistic deterministic tree automata for the same corpus. According to the results, we can say that the classification is improved splitting the melody in bars. Also the results keep in good performance taking the string of the whole melody, which is important since not always the bar information is available. We are persuaded that these promising results can be improved by defining a more complex universal grammar for unseen labels and removing some rules that make the grammars more ambiguous.

Thus, it seems that tree grammars are able to classify noisy snippet queries. Although the success rate seems to increase with the query length we need find a balance between success and computation time. Moreover, the ranking could be used as a pruning for discarding the parse of a whole melody for the worse grammars given the snippet query. In other words, if we have a long melody, we could classify it using only its beginning. After that, we compute the probability of the whole melody only for the best classes. This way, we can improve the computation time when the melody length is large.

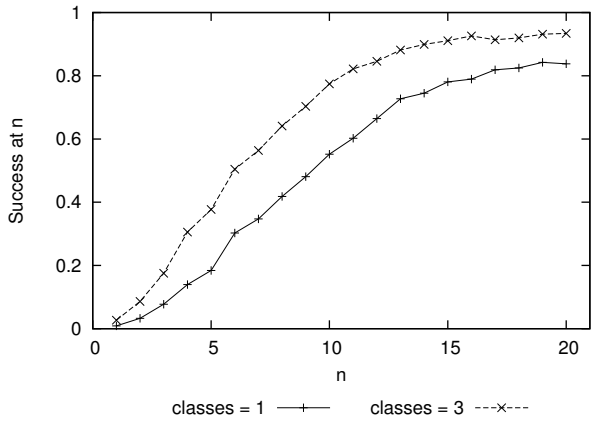


Figure 6.10: Success rate when the proper class is in the best i classes (*Essen-Kinder*).

7

Conclusions and Future Work

7.1 Conclusions

In this thesis, the problem of music similarity has been studied. In order to disambiguate this concept for the span of this dissertation, we had considered as ground-truth that the most similar sequences are different interpretations of the same song or those produced by the variation compositional form. This statement implies that the similarity is measured upon a trade-off between the melodic and rhythmic dimensions. To perform our experiments, we have focused on a standard task in music information retrieval, namely melody classification. The approaches used try to emulate this classification in a similar way to how humans do, recognizing previously known patterns, even if the inputs are distorted, they are presented just partially, or in the presence of noisy data.

The research has been done in the symbolic music data field using a tree representation named *metrical trees* and described in (Rizo, 2010). The tree structure is an alternative representation between strings and graphs. On the one hand, its expressive capacity is higher than the strings and allow describing naturally hierarchical structures in which relations among its components are given. On the other hand, its management from the theoretical point of view is much simpler and efficient than that of graphs. In this structure, leaves contain pitches. Duration is represented by the level of the tree: the shorter a note is, the deeper it appears in the tree and an in-depth tree traversal represents the time sequence of music. This representation has proven to be effective in a number of melodic similarity computation in tasks (Habrard et al., 2008; Rizo, 2010).

Tree edit similarity learning Our first contribution has been focused on the tree similarity learning problem. We investigated a new framework for learning tree edit distances thanks to a convex optimization problem based on

the framework called *GESL* for *Good Edit Similarity Learning* and described in (Bellet et al., 2012), originally developed for strings. We have shown that this framework is adequately tailored to tree edit distance allowing us to have strong theoretical justification while having an efficient procedure to deal with complex distance, such as the Zhang-Shasha one, which is a clear advantage in comparison of EM-based methods that quickly become intractable.

We experimentally showed on the music recognition task described above that this framework is able to build very accurate classifiers improving state-of-the-art results for this problem. This experiment was done in a multi-class setting showing that GESL can also deal with this context. Moreover, we illustrated that the produced models can be used to provide a semantic analysis of the knowledge learned from the data.

Stochastic k-testable tree automata Our second contribution has been focused on the inference of k-testable tree languages showing that they are suitable for the classification of tree-represented music data. For that, the probabilities of several inferred stochastic k-testable tree automata (one for each class) to generate a given tree (melody) were used to classify the given tree in the class that maximizes this probability.

The results overcame those previously obtained using tree edit distances for the same corpora even the obtained with the tree edit distance learning approach using linear classifiers described in Chapter 4, as we show in Table 7.1

Table 7.1: Results summary for success rates (%) and comparison with the tree edit distance learning approach using linear classifiers and k-testable tree automata for the (*Pascal*) corpus.

Approach	Success rate Linear Classifier
K_{Selkow}	93.1 ± 0.5
$K_{Zhang-Shasha}$	95.0 ± 0.7
	Tree automata
PDTA $K = 3$	95.1 ± 0.2
PDTA $K = 4$	96.1 ± 0.6

Based on the results we can say that the classification is improved when taking into account all the bars of the melody rather than using them separately. We can therefore say that stochastic k-testable tree models

somehow capture the structure of the melody, so that this structure could be used to improve the melody classification.

Another important point here is efficiency and the scalability of the approach. It is remarkable that the classical tree edit distance has a high complexity, $O(m_A m_B \max\{h_A, h_B\})$, where m_i are the maximum arities of the two trees, and h_i their heights, which in practice is actually $O(|M_A||M_B| \max\{h_A, h_B\})$. This must be multiplied by the complexity (quadratic) of the nearest neighbour method implemented to perform the classification, so if $n = |\Sigma|$ the total complexity is $O(n^2|M_A||M_B| \max\{h_A, h_B\})$. On the other hand, the proposed method has been shown (García, 1993) to run in $O(|M|^{k-1} n \log n)$ time and the classification is performed in linear time with $|t|$, once k is fixed. In addition, stochastic k-testable tree-models can be updated with new samples and do not require too many training samples to get an acceptable success rate as seen in the results.

Probabilistic tree grammars Our third contribution has been focused on the application of probabilistic tree grammars constructed from stochastic k-testable tree-models showing that this approach can be used for classifying new melodies represented by strings using the information captured in the grammar rules. This approach allows avoiding the duration information in the input data (strings with pitch only), making easier querying a music database.

Symbolic music retrieval from queries has been extensively studied in the MIR literature. In order to search a query in a dataset of songs, we can apply any of the well-known pattern matching algorithms, like local string editing to each of the songs in the dataset, and retrieve a ranking of most similar items. The main problem here is the efficiency when using large datasets, but with the advantage that it can find a partial or approximate occurrence of the query in any part of the dataset. On the other hand, a previous indexation of the dataset, e.g by means of motive extraction, solves the scalability issue, but motive extraction usually needs to work with exact repetitions, making very inaccurate to build robust indices from different interpretations of the same song. In addition, when searching only in motives, the music not present in the motives is hidden.

Our proposal is able to solve intrinsically these problems. The probabilistic grammar structure itself encodes both motives and melody variations, giving more weight to the most repeated themes, without the need to be exact, and enabling the possibility to learn from different renderings of the same song, leaving aside the need to encode motives. Besides, there is no

difference between looking for a whole song or searching a small query in the dataset.

The results overcame those previously obtained using probabilistic deterministic tree automata for the same corpus. According to the results, we can say that the classification is improved splitting the melody in bars. Also the results keep in good performance taking the string of the whole melody, which is important since not always the bar information is available.

Thus, it seems that tree grammars are able to classify noisy snippet queries. Although the success rate seems to increase with the query length we need find a balance between success and computation time. Moreover, the ranking could be used as a pruning for discarding the parse of a whole melody for the worse grammars given the snippet query. In other words, if we have a long melody, we could classify it using only its beginning. After that, we compute the probability of the whole melody only for the best classes. This way, we can improve the computation time when the melody length is large.

Generative system Finally, our last contribution is derived from the inference of the tree grammars described in Chapter 6. These grammars can be used to classify melodies as we showed above but also could be used to generate new melodies using the structure of the trees captured in the different grammar rules inferred from the tree data. In this way, we could build a system that would allow us to generate new melodies from the inferred grammars.

This system could be used in several ways. First, the possibility to generate genuine music from several databases like genre, style or author. In this way, the new music would have the essence of the dataset captured in the grammar. Second, the mentioned system could be used to help music composition systems suggesting new ideas, derived from the several training sets, to write new songs. Third, this generation engine could be used in situations that requires obtain more samples automatically. For example, in databases that have limited samples and can not be used with other pattern recognition approaches because them require a huge number of samples. Eventually, in any situation that requires an automatic generation of new samples.

Unfortunately, the construction of a generative system requires a deeper research which is out of the scope of this thesis. However, we show some lines to follow and an early research with some ideas in the future work section.

7.2 Future works

There are a number of possible lines of work that can be followed to study more in depth the approaches proposed in this thesis. Some of them are related to the tasks presented here and others can be considered a continuation of this work by applying the same methods to other music information retrieval tasks.

First, the several approaches proposed in this thesis could be used to other kind of classes like genre, style or author. To do this, we only need new corpora properly labelled to apply the methods studied. An early research was done using the corpus Perez-9-genres described in (Pérez-Sancho, 2009) where the task consists in classify melodies in different music genres. The preliminary results seem to be hopeful.

Second, regarding the edit similarity learning approach described in Chapter 4, a perspective of this work would be to study other definitions of tree edit similarities. Indeed, GESL is based on a linear combination of the edit script operations plugged in an exponential but we could imagine other strategies tailored to the application at hand. Another interesting future work would be to adapt the similarity learning procedure directly to the multi-class setting instead of binary classification. Moreover, the efficiency of this tree edit distance learning framework (both in terms of accuracy and running time) opens the door to an extensive use of tree edit distance in larger-scale applications such as natural language processing or XML data classification.

Third, about the k-testables tree automata and grammars. We are persuaded that the promising results obtained could be improved by adjusting some parameters not studied yet, like using different and more sophisticated discount methods. In other way, results and efficiency could be improved by defining a more complex universal grammars (automata) for unseen labels and removing some rules that make the grammars (automata) more ambiguous.

Finally, as we pointed out above, in Section 7.1, the tree grammars described in Chapter 6 can be used to classify melodies as we showed above but also could be used to generate new melodies using the structure of the trees captured in the different grammar rules inferred from the tree data. In this way, we could build a system that would allow us to generate new melodies from the inferred grammars.

This system can be constructed using a randomly approach to generate new melodies from the probabilities captured in the grammar rules. In other words, a new melody is generated using the random walk formalization.

Starting from the Start symbol, a new rule is randomly selected in each step. The process finish when all the current symbols are terminals. At this point, a new tree (melody) is obtained.

After having done some preliminary tests, some problems have appeared. One of them is that the generated trees can be very deep and have few bars. An early solution to solve this problem can be select the trees according depth and number of bars. Other solution can be change the tempo according the depth of the tree.

Another way to generate the new melodies could be generate strings of pitches whit a genetic algorithm and then parse this string with the tree grammars retrieving the most likelihood parse tree from a string of pitches. Therefore, a new melody is obtained with the pitches generated from the genetic algorithm and the tree structure obtained from the tree grammars.

Regarding the problem of how evaluate the generated melodies, a possible idea is to use a edit similarity learned with the approach described in Chapter 4 which is totality independent of the inferred grammars. In this way, we could have an indicator to evaluate the generated melodies.

Finally, after observing the preliminary results of the generated melodies seems the models are bar-oriented. This fact was also perceived in tree automata which was solved using the dynamic time model described in Section 5.3.2. However, it seems to be suitable adding more information about the whole structure. Therefore, this fact open the door to study the incorporation of harmonic information to the tree structure. A possible solution is generate an harmonic sequence and then generate each tree bar of the sequence until the whole melody is completely generated. Unfortunately, this is another story and we do not know if it will be solved in a brief time slice.

A Corpora

A.1 Pascal corpus

- "Alouette" (french children song)
- "Macarena" (Los del Rio)
- "Avemaria" (F. Schubert) (Fig. A.1 contains the score of the used theme and Fig. A.2 for an example of some interpretations of that theme).
- "Ode to joy, from the 9th symphony" (L.V. Beethoven)
- "Boléro" (M.Ravel)
- "Oh! Susanna" (S. Foster)
- "La Cucaracha" (mexican song)
- "Pink Panther" (H. Mancini)
- "La Cumparsita" (G.M. Rodriguez)
- "Silent night" (christmas carol)
- "Frére Jacques" (french children song)
- "Tico-Tico no Fubá" (Zequinha de Abreu)
- "Guantanamera" (José Fernández Díaz)
- "Toccata and fugue in D minor" (J.S.Bach)
- "Happy birthday" (Patty Hill and Mildred J. Hill)
- "Twinkle twinkle little star" (children song)

- "Jingle bells" (J. Pierpoint)
- "When the Saints Go Marching In" (J.M. Black)
- "Lohengrin, wedding march" (R. Wagner)
- "Yesterday" (The Beatles)

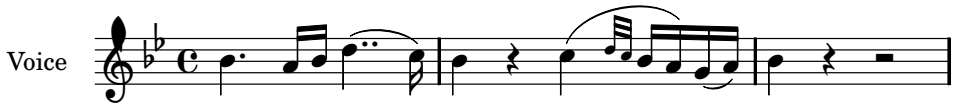
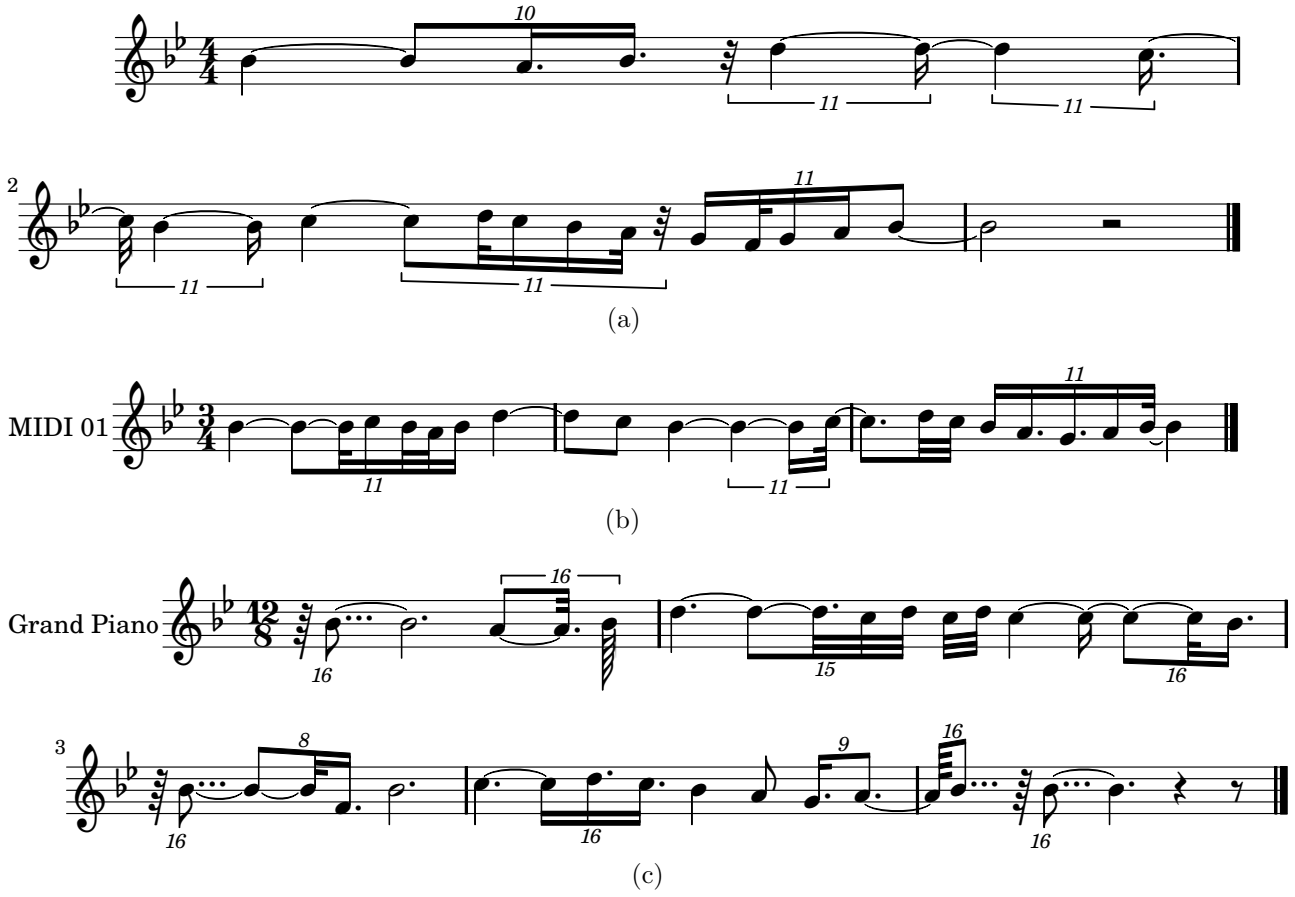


Figure A.1: Incipit of the Schubert's Ave Maria used as query.



(a)

(b)

(c)

Figure A.2: Three different interpretations of the Schubert's Ave Maria used for training (scores rendered using the MakeMusic Inc. Finale package from the MIDI files).

A.2 ESSEN synthetic corpus example

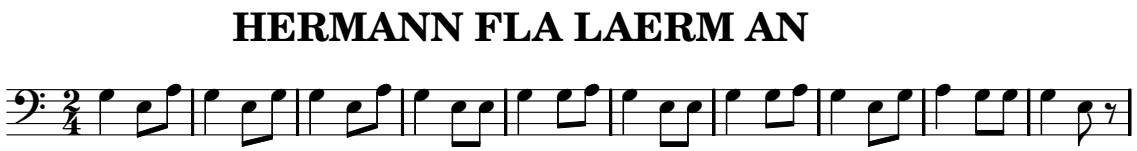


Figure A.3: Theme K1605 as present in the ESSEN corpus.

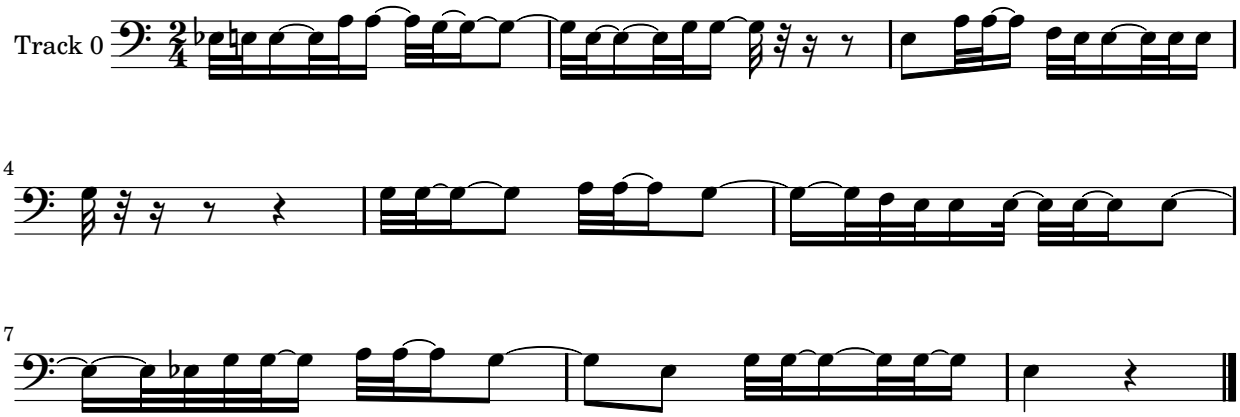


Figure A.4: Distorted theme from melody shown in A.3 (score rendered using the MakeMusic Inc. Finale package from the MIDI file).

B
Publications

Some parts of this thesis have been published in journals and conference proceedings. Here is a list of papers in chronological order:

- Bellet, A; Bernabeu, J.F.; Habrard, A.; Sebban, M
  "Learning discriminative tree edit similarities for linear classification - Application to melody recognition"
  Neurocomputing, vol. 214, pp. 155-161 (2016)

- Bernabeu, J. F.; Calera-Rubio, J.; Iñesta, J. M.; Rizo, D.
  "Query Parsing Using Probabilistic Tree Grammars"
  5th workshop on Music and Machine Learning, MML 2012, Edinburgh (2012)

- Bernabeu, J.F.; Calera-Rubio, J.; Iñesta, J.M.
  "Classifying melodies using tree grammars"
  Pattern Recognition and Image Analysis: 5th Iberian Conference, IbPRIA 2011, Las Palmas de Gran Canaria, Spain, June 8-10, 2011.
  Lecture Notes in Computer Science, vol. 6669, pp. 572–579 (2011)

- Bernabeu, Jose F. and Calera-Rubio, Jorge and Iñesta, José M. and Rizo, David
  "Melodic Identification Using Probabilistic Tree Automata"
  Journal of New Music Research, vol. 40, pp. 93-103 (2011)

- Bernabeu, J.F., Calera-Rubio, J., Iñesta, J.M., Rizo, D.
  "Tree language automata for melody recognition"
  Actas del II Workshop de Reconocimiento de Formas y Anlisis de Imgenes (AERFAI), ISBN: 978-84-92812-66-0, pp. 17-22, Valencia, Spain (2010)

- Bernabeu, J.F., Calera-Rubio, J., Iñesta, J.M., Rizo, D.
  "A probabilistic approach to melodic similarity"
  Proceedings of MML 2009, pp. 48-53 (2009)

C

Resumen en castellano

Una de las principales preocupaciones en las tareas de recuperación de información musical (MIR) es cómo evaluar la similitud melódica de una manera similar a cómo lo hacen los seres humanos. El ser humano es capaz de reconocer patrones previamente conocidos, incluso si lo que perciben nuestros sentidos está distorsionado, se presentan sólo parcialmente, o los datos se presentan en un ambiente ruidoso. En la comparación musical esto sucede en varias situaciones, por ejemplo, al comparar versiones de una melodía dada o al buscar en bases de datos utilizando una consulta que será, por definición, parcial y puede ser distorsionada o incluso errónea. Dos cuestiones se refieren a este problema: el cálculo de similitud y la estructura de representación utilizada.

El término similitud musical es ambiguo o al menos puede ser juzgado desde diferentes puntos de vista (Selfridge-Field, 1998). Puede referirse a la semejanza entre la línea melódica de dos fragmentos musicales, la similitud de sus patrones rítmicos, o incluso su coincidencia armónica. Con el fin de desambiguar este concepto para el contexto de esta tesis, consideraremos como verdad fundamental que las secuencias más similares son interpretaciones diferentes de la misma canción o de las producidas por la variación de la composición. Esta afirmación implica que la similitud es medida a partir de un equilibrio entre la dimensión melódica y rítmica.

C.1 Recuperación de información musical (MIR)

La recuperación de información musical (MIR) es un campo de investigación dedicado a la extracción de información significativa a partir del contenido de fuentes musicales (Orio, 2006), (Typke et al., 2005). Tradicionalmente, esta investigación se ha dividido en dos dominios: audio y simbólico. Los archivos de audio digital contienen una señal de audio digitalizada procedente de una grabación de sonido y se pueden encontrar en formatos comprimidos (MP3)

o sin comprimir (WAV). Por otro lado, los archivos de música simbólica contienen partituras digitales, es decir, la notación musical y las instrucciones necesarias para que una computadora o un sintetizador reproduzca una canción. Estos archivos pueden ser secuenciados (MIDI) o estructurados (MusicXML).

Con el uso generalizado de reproductores de música portátiles, la mayor parte de los usuarios de ordenadores de hoy en día tienen que lidiar con una gran base de datos de música digital. Muchas herramientas de software (por ejemplo, iTunes) están disponibles para reproducir, recuperar y almacenar grandes cantidades de archivos de audio. Estos archivos contienen una información más rica que la información proporcionada por datos simbólicos, ya que reúnen todos los elementos que participan en una obra musical: el tono, la armonía, el ritmo, el timbre, las letras, etc. Sin embargo, esta riqueza da lugar a una complejidad mayor, porque toda esta información está mezclada con la señal de audio y es muy difícil separar cada uno de estos elementos. Para trabajar con archivos de audio es necesario utilizar técnicas de procesamiento de señales digitales para extraer características que representen el contenido musical. Por ejemplo, una de las características más comunes utilizadas para este propósito son los coeficientes cepstrales en las frecuencias de Mel, que proporcionan información sobre el timbre de la pieza musical (Aucouturier y F., 2004). Otras obras utilizan características relacionadas con el ritmo (Lidy y Rauber, 2005), textura (Tzanetakis y Cook, 2002), o tono (Tzanetakis et al., 2003). Sin embargo, estas características sólo proporcionan descripciones poco precisas del contenido musical y son difíciles de interpretar.

Debido a la repentina y enorme popularidad de los reproductores de audio portátiles, la investigación sobre algoritmos que clasifican, recuperan y recomiendan música se han vuelto realmente importante en los ultimos tiempos. En este marco, surge la necesidad de transformar los archivos de audio en un formato que proporcione información muy precisa sobre el contenido real de una obra musical - la partitura - incluyendo tono, armonía, ritmo y letras, pero que carecen de algunas características importantes que sóolo pueden encontrarse en los archivos de audio, como el timbre, interpretacióon y características de la producción. Los archivos que obtienen este tipo de características son los archivos de música simbólica (datos), y el proceso para transformar datos de audio en datos simbólicos se denomina transcripción. Sin embargo, los algoritmos de transcripción de última generación no son fiables hoy en día. Por lo tanto, la mayoría de los modelos musicales de hoy en día no consideran la estructura musical en absoluto. En su mayoría dependen de las propiedades locales de la seal de audio, como la textura, o el análisis de frecuencia a corto plazo. Por ejemplo, en la

mayoría de los enfoques actuales para la transcripción o seguimiento de tono, los algoritmos tienen que basarse en fuertes suposiciones sobre el timbre o el número de notas simultáneas para decidir cuántas notas se escuchan (o interpretan) simultáneamente, e identificar estas notas. Por lo tanto, la aplicabilidad general de estos algoritmos es limitada.

A pesar de ello, se ha hecho muy poca investigación para modelar datos de música simbólica en comparación con los importantes esfuerzos desplegados para modelar datos de audio. Hay algunos trabajos que utilizan datos simbólicos para mejorar los resultados utilizando sólo datos de audio. Un ejemplo de esto es el enfoque en el contexto de la clasificación de género propuesto por (Lidy T., 2007) donde las características de audio y características simbólicas se combinan para conseguir una mejora en los resultados de clasificación.

Otro ejemplo es el trabajo de (Paiement, 2008) en el que modelos precisos de música simbólica podrían mejorar drásticamente el rendimiento de los algoritmos de transcripción aplicados en contextos más generales. Proporcionarían "conocimiento musical" a algoritmos que actualmente sólo dependen de las propiedades básicas del sonido para tomar decisiones. Del mismo modo, los modelos de lenguaje natural se usan normalmente en los algoritmos de transcripción del habla (Rabiner y Schafer, 1978). Como un ejemplo simple, supongamos que un algoritmo de transcripción conoce la clave de una canción en particular y trata de estimar la última nota de esta canción. La probabilidad de que esta nota fuese la tónica sería muy alta, ya que la mayoría de las canciones en cualquier corpus terminan en la nota tónica. Otra ventaja de los datos de música simbólica es que son mucho más comprimidos que los datos de audio. Por ejemplo, la representación simbólica de un archivo de audio de docenas de megabytes puede transformarse tan sólo en una pequeña cantidad de kilobytes. Esta pequeña cantidad de kilobytes contiene la mayor parte de la información necesaria para reconstruir el archivo de audio original. De este modo, podemos concentrarnos en las características psicoacústicas esenciales de la señal al diseñar algoritmos para capturar dependencias a largo plazo en datos de música simbólica. Finalmente, la ventaja más interesante de tratar directamente con datos simbólicos es la posibilidad de diseñar algoritmos de generación de música realistas. La mayoría de los modelos probabilísticos presentados en esta tesis son modelos generativos. Por lo tanto, estos modelos pueden ser muestreados para generar eventos musicales genuinos, dados otros componentes musicales o no. Sin embargo, la generación de nueva música está fuera del alcance de esta tesis.

C.2 La dimensión melódica y rítmica

Dependiendo del dominio de aplicación, se dice que el contenido musical contiene diferentes atributos. En el dominio psicoacústico, (Levitin, 1999) utiliza el tono, el ritmo, el tempo, el contorno, el timbre, la sonoridad y la localización espacial. En el dominio MIR, (Downie, 1999b) considera siete dimensiones: tono, temporal, armónica, timbral, editorial, textual y bibliográfica. Otros autores también incluyen características más elaboradas como la información temática obtenida a partir de los datos brutos (Hsu et al., 1998). Para la comparación basada en contenido musical, las propiedades más utilizadas y directamente disponibles son el tono y el ritmo. En esta tesis se utilizarán estos dos atributos para describir las notas musicales, por lo que, es conveniente que el lector conozca su definición.

C.2.1 Altura o tono

El tono es la frecuencia percibida de un sonido (Krumhansl, 1979). El contenido de la frecuencia de una nota musical se compone de una frecuencia fundamental y múltiplos enteros de esa frecuencia. La percepción del tono humano es logarítmica con respecto a la frecuencia fundamental. Por lo tanto, normalmente nos referimos al tono de una nota usando clases de tono. En inglés, una clase de tono se define por una letra. Por ejemplo, la nota con la frecuencia fundamental de 440 Hz se llama A (LA en castellano). En la cultura de la música occidental, la escala cromática es el método más común de organizar las notas. Cuando se utiliza la escala temperada, cada nota sucesiva está separada por un semitono. Dos notas separadas por un semitono tienen una relación de frecuencia fundamental de $2^{1/12}$ (aproximadamente 1.05946). Usando este sistema, la frecuencia fundamental se duplica cada 12 semitonos. El intervalo entre dos notas se refiere al espacio entre estas dos notas con respecto al tono. Se dice que dos notas separadas por 12 semitonos están separadas por una octava, y tienen la misma clase de tono. Por ejemplo, la nota con frecuencia fundamental a 880 Hz se llama A, una octava más alta que la nota A con la frecuencia fundamental a 440 Hz. Decimos que el intervalo entre estas dos notas es una octava. Observe que la nota correspondiente a 440 Hz es A4 y A5 corresponde a 880 Hz mientras que su clase de tono es la misma, en este caso A. El símbolo '♯' (sostenido) eleva una nota en un semitono. Por el contrario, el símbolo '♭' (bemol) baja una nota en un semitono. La mayoría de las clases de tono están separadas por un tono (es decir, dos semitonos), excepto para las notas E y F, así como B y C, que están separados sólo por un semitono. En este sistema, $A^{\flat}$ y $B^{\flat}$ se refieren a la misma nota. Dos clases de tono que se refieren a la misma

altura se les denomina enarmónicas. En esta tesis, consideraremos que las notas enarmóonicas son completamente equivalentes.

C.2.2 Ritmo

En la mayoría de las notaciones musicales, el ritmo se define con respecto a un pulso subyacente que divide el tiempo en partes iguales. La velocidad del ritmo se llama tempo. Por ejemplo, cuando el tempo es 120, contamos 120 pulsos por minuto (BPM), o dos pulsos por segundo. La métrica es el sentido de los pulsos fuertes y débiles que surge de la interacción entre niveles jerárquicos de secuencias que tienen componentes periódicos anidados. Tal jerarquía está implícita en la notación de música occidental, donde los diferentes niveles están indicados por tipos de notas (notas redondas, notas blancas, notas negras, etc.) y donde los compases establecen segmentos que contienen igual número de pulsos (Handel, 1989). Los tipos de notas se definen relativamente entre sí. Las notas redondas tienen siempre el doble de duracióon que las blancas, que a su vez tienen el doble de la duracióon que las negras, y así sucesivamente. El número de pulsos por compás se define generalmente en el principio de una canción por la indicación de compás. El numerador representa el número de tiempos o pulsos que tendrá el compás. El denominador representa la unidad de pulso, que es la figura que llenará un tiempo del compás. Por ejemplo, en la mayoría de los compases de cuatro pulsos, una nota negra dura un tiempo. Por lo tanto, una nota corchea dura la mitad de un pulso, una blanca dura dos pulsos, y una redonda dura cuatro pulsos. Si el tempo es 120, interpretaremos una blanca por segundo y hay dos blancas en cada compás.

C.3 La representación de árbol

Una serie de trabajos han abordado el problema de cómo representar la música simbólica de manera que la tarea de comparación pueda ser eficaz y eficiente. La eficacia se puede medir en términos de su capacidad para hacer frente a diferentes aspectos de la percepción humana de similitud melódica. Esta es una tarea difícil y se basa principalmente en los datos contenidos en los conjuntos de entrenamiento de los experimentos realizados y los algoritmos de comparación. Por otro lado, la eficiencia se relaciona principalmente con las complejidades de tiempo y memoria. Este aspecto evalúa hasta qué punto el enfoque considerado puede ser escalado a situaciones del mundo real con cientos o incluso miles de datos musicales. Las melodías musicales han sido comparadas por diferentes algoritmos que

utilizan diferentes tipos de representaciones. Codificaciones de cadenas junto con distancias de edición y alineación usando numerosas representaciones de tono y ritmo fueron estudiadas en (Grachten et al., 2005, Lemger m, 2000, Mongeau y Sankoff, 1990) y algoritmos basados en modelos de n-gramas en (Doraisamy , 2004, Downie, 1999a, Uitdenbogerd, 2002). Además, se propuso una codificación de grafos en (Pinto y Tagliolato, 2008) en la que la información del ritmo no estaba representada. Otros enfoques para la representación son menos abstractos y tratan de asignar los tonos de melodía y las duraciones en graficos 2D, en una especie de diagráma de pianola, convirtiendo el problema de coincidencia de melodías en uno geométrico (Aloupis et al., 2006, Tanur, 2005, Typke, 2007 , Ukkonen et al., 2003, Wiggins et al., 2002). La estructura de árbol es una representación alternativa entre cadenas y grafos. Por una parte, su capacidad expresiva es superior a las cadenas y permite describir estructuras naturalmente jerárquicas en las que se dan las relaciones entre sus componentes. Por otra parte, su gestión desde el punto de vista teórico es mucho más simple y eficiente que la de los grafos. Los árboles han sido utilizados por varios autores en la literatura con diferentes objetivos. La teoría del lenguaje formal usa los árboles de una manera natural y Lee (Lee, 1985) intentó interpretar ritmos usando gramáticas. Algo similar hizo Bod (Bod, 2002), pero con el objetivo de aprender a segmentar automáticamente melodías, utilizando el enfoque de árbol proporcionado por el análisis de la melodía. En el trabajo de Conklin (Gilbert y Conklin, 2007), melodias monofónicas fueron transformadas en estructuras de árbol utilizando una gramática libre de contexto probabilística para realizar reducciones melódicas que se utilizaron también en una tarea de segmentación.

En el contexto de la composición musical asistida, los árboles se han utilizado como una forma de representar conceptualmente la música (Balaban, 1996; Smaill et al., 1993). Bajo este enfoque, el sistema Wind in the Willows (Hogberg, 2005) usó transductores de árboles para generar música. La renombrada herramienta para la composición asistida OpenMusic (Assayag et al., 1999) utiliza árboles como una forma natural para representar la naturaleza jerárquica de la duración de la subdivisión de las figuras musicales y agrupaciones como tuplas. Para representar los análisis musicológicos, la representación de los árboles se utilizó en la Teoría Generativa de la Música Tonal (GTTM) (Lerdahl y Jackendoff, 1983) y en una serie de trabajos basados en el análisis de Schenker (Kirlin y Utgoff, 2008; Marsden, 2001, 2005; , 2007, Smoliar, 1979). Por último, los árboles también se han utilizado no como un medio para representar la música, sino como una estructura de datos intermedia para otros objetivos, como la construcción de estructuras documentos para la indexación (Blackburn,

2000; Skalak et al., 2008). En este trabajo se ha empleado una representación de árbol que codifica las melodías, la cual fue introducida en (Rizo, 2010; Rizo et al., 2003) donde se utiliza la estructura de árbol para codificar el ritmo implícitamente. En esta estructura, las hojas contienen las alturas de las notas. La duración está representada por el nivel del árbol: cuanto más corta es una nota, más profunda aparece en el árbol y un recorrido en profundidad del árbol representa la secuencia temporal de la música. Se ha demostrado que esta representación es eficaz en una serie de tareas en las que se realizaban cálculos de similitud melódica (Habrard et al., 2008; Rizo, 2010).

Una de las principales ventajas de los árboles melódicos es que, a diferencia de las representaciones lineales en las que ambas dimensiones melódicas: tiempo (duración) y tono están codificadas por símbolos explícitos, representan implícitamente el tiempo en su estructura, haciendo uso del hecho de que las duraciones de nota son múltiplos de unidades de tiempo básicas. De esta manera, son menos sensibles a la codificación de la melodía, ya que sólo se necesita codificar el tono, por lo que hay menos grados de libertad para la codificación y, por lo tanto, menos parámetros a ajustar. Sin embargo, esta representación tiene algunos inconvenientes: su estricta dependencia de la estructura de la métrica de la fuente de entrada y su dificultad para representar ligaduras, puntillos y síncopas. El primer problema puede ser resuelto a través de un análisis métrico a priori (Eck y Casagrande, 2005, Meudic, 2002) en el caso de que los metadatos del medidor no estén presentes en la fuente de datos. El segundo inconveniente, desde el punto de vista de la representación, puede resolverse mediante la adición de un símbolo especial que codifica el concepto de continuación de nota. Para la tarea de comparación, esto no es un problema como otros autores señalan (Hanna et al., 2008, Mongeau y Sankoff, 1990, Pardo y Sanghi, 2005). Un problema derivado es el crecimiento excesivo de árboles cuando se encuentran notas muy cortas y imprecisiones de rendimiento en el caso de datos secuenciales en tiempo real. Este problema ha sido abordado por métodos de poda de árboles y algoritmos avanzados de cuantificación (Agon et al., 1994, Cemgil et al., 2000). Finalmente, los métodos para comparar este tipo de estructuras son costosos, en algunos casos se vuelven casi intratables y se hace necesario utilizar métodos que sean más eficientes.

C.4 Objetivos y enfoque en esta tesis

Debido a las ventajas de la estructura de árbol descrita anteriormente, nuestra motivación para realizar esta tesis consiste en intentar construir un

sistema que permita clasificar y generar melodías utilizando la información de la codificación del árbol, capturando las dependencias inherentes que se encuentran dentro de este tipo de estructura. Y mejorar los métodos actuales en términos de precisión y tiempo de ejecución. De esta manera, trataremos de encontrar métodos más eficientes que son clave para usar la estructura de árbol en grandes conjuntos de datos. En primer lugar, estudiaremos las posibilidades de la distancia de edición de árboles para clasificar melodías utilizando un nuevo enfoque para estimar los pesos de las operaciones de edición. Centrándonos en el problema de aprendizaje de similitud de árboles, investigaremos un nuevo marco para el aprendizaje de las distancias de edición de árboles gracias a un problema de optimización convexa basado en el marco denominado GESL (Good Edit Similarity Learning) descrito en (Bellet et al., 2012).

Una vez estudiadas las posibilidades del enfoque citado, se utilizará un enfoque alternativo. Para ello se utilizará un enfoque de inferencia gramatical para inferir lenguajes de árboles. La inferencia de estos lenguajes nos da la posibilidad de utilizarlos para clasificar nuevos árboles (melodías). Por otra parte, este enfoque podría ser utilizado para generar nuevas muestras, aunque esta cuestión está fuera del alcance de esta tesis. Para llevar a cabo esta investigación nos centramos en dos enfoques diferentes. Por un lado, el formalismo de los autómatas de árboles es utilizado para inferir los modelos y clasificar las melodías representadas como árboles. Para ello, se utilizarán los lenguajes k-testables de árboles. Por otro lado, las gramáticas de árboles inferidas a partir de los autómatas de árboles. De esta manera, las gramáticas inferidas usando datos de árbol en el entrenamiento se utilizarán para clasificar nuevas melodías representadas como cadenas. Este el último enfoque es necesario para resolver el problema cuando la información de duración de las notas no está disponible en las muestras que tienen que ser clasificadas.

C.5 Conclusiones

En esta tesis se ha estudiado el problema de la similitud musical. Para desambiguar este concepto a lo largo de esta disertación, habíamos considerado como verdad fundamental que las secuencias más parecidas son las diferentes interpretaciones de una misma canción o de las producidas por la variación en la composición. Esta afirmación implica que la similitud se mida a partir de un equilibrio entre la dimensión melódica y rítmica. Para realizar nuestros experimentos, nos hemos centrado en una tarea estándar en la recuperación de información musical, esta es, la clasificación de la

melodía. Los enfoques utilizados tratan de emular esta clasificación de manera similar a cómo lo hacen los seres humanos, reconociendo patrones previamente conocidos, incluso si los datos de entradas están distorsionados, se presentan sólo parcialmente, o se presentan junto a ruido. La investigación se ha realizado en el campo de datos musicales simbólicos utilizando una representación de árbol denominada árboles métricos y descrita en (Rizo, 2010). La estructura del árbol Es una representación alternativa entre cadenas y grafos. Por un lado, su capacidad expresiva es superior a las cadenas y permiten describir estructuras naturalmente jerárquicas en las que sus componentes pueden ser relacionados. Por otra parte, su gestión desde el punto de vista teórico es mucho más simple y más eficiente que la de los grafos. En dicha estructura, las hojas contienen las alturas de las notas. La duración es representada por el nivel del árbol: cuanto más corta es una nota, más profunda aparece en el árbol y un recorrido en profundidad del árbol representa la secuencia temporal de la música. Se ha demostrado que esta representación es eficaz en una serie de tareas en las que se realizaban cálculos de similitud melódica (Habrard et al., 2008; Rizo, 2010).

Aprendizaje de similitud de árboles Nuestra primera contribución se ha centrado en el problema del aprendizaje de similitud de árboles. Se ha investigado un nuevo marco para el aprendizaje de las distancias de edición de árboles gracias a un problema de optimización convexa basado en el marco denominado GESL (Good Edit Similarity Learning) descrito en (Bellet et al., 2012), originalmente desarrollado para cadenas. Hemos demostrado que este marco se puede adaptar adecuadamente a las distancias de edición de árboles, además de ser un procedimiento eficiente para hacer frente a distancias complejas, como puede ser el caso de la distancia de árboles de Zhang-Shasha, lo cual es una clara ventaja en comparación con los mtodos basados en EM (esperanza-maximización) que se vuelven rápidamente intratables. Hemos demostrado experimentalmente que, en la tarea de reconocimiento de la música descrita anteriormente, este marco es capaz de construir clasificadores muy precisos que mejoran los resultados actuales para este problema. Este experimento se realizó en un entorno multi-clase mostrando que GESL también puede hacer frente a este contexto. Además, hemos ilustrado que los modelos producidos pueden ser utilizados para proporcionar un análisis semántico del conocimiento aprendido de los datos.

Autómatas probabilísticos k-testables de árboles Nuestra segunda contribución se ha centrado en la inferencia de lenguajes k-testables de

árboles mostrando que son adecuados para la clasificación de los datos musicales representados como árboles. Para ello, se utilizaron las probabilidades de varios autómatas probabilísticos k-testables (uno para cada clase) de generar un determinado árbol (melodía) para clasificar el árbol dado en la clase que maximiza esta probabilidad.

Los resultados superaran los obtenidos previamente usando las distancias de edición de árbol para los mismos datos, incluso los obtenidos con las distancias aprendidas mencionadas anteriormente utilizando los clasificadores lineales, como se muestra en la Tabla 7.1. Basándonos en los resultados podemos decir que la clasificación es mejorada cuando se tienen en cuenta tods los compases de la melodía en lugar de utilizarlos por separado. Por lo tanto, podemos decir que los modelos de árboles estocásticos k-testables capturan de alguna manera la estructura de la melodía, de modo que esta estructura puede usarse para mejorar la clasificación de la melodía.

Otro punto importante aquí es la eficiencia y la escalabilidad del enfoque. La distancia de edición clásica del árbol tiene una alta complejidad, $O(m_A m_B \max\{h_A, h_B\})$, donde m_i son las aridades máximas de los dos árboles, y h_i, sus alturas, que en la práctica es realmente $O(|M_A||M_B| \max\{h_A, h_B\})$. Esto debe ser multiplicado por la complejidad (cuadrática) del método de vecino más cercano implementado para realizar la clasificación, por lo que si $n = |\Sigma|$, la complejidad total es $O(n^2|M_A||M_B| \max\{h_A, h_B\})$. Por otra parte, el método propuesto se mostró en (García, 1993) que tenía una complejidad temporal de $O(|M|^{k-1} n \log n)$ y la clasificación se realiza en tiempo lineal con $|t|$, una vez que k es fijo. Además, los modelos estocásticos k-testables de árboles pueden ser actualizados con nuevas muestras y no requieren demasiadas muestras de entrenamiento para obtener una tasa de acierto aceptable como se muestra en los resultados.

Gramáticas de árboles probabilísticas Nuestra tercera contribución se ha centrado en la aplicación de las gramáticas de árboles construidas a partir de modelos estocásticos k-testable de árboles mostrando que este enfoque puede ser utilizado para la clasificación de nuevas melodías representadas por cadenas utilizando la información capturada en las reglas gramaticales. Este enfoque permite evitar el problema cuando la información de duración de notas no está disponible en los datos de entrada (cadenas con tono solamente), facilitando la consulta de una base de datos musical. La recuperación de música simbólica mediante consultas ha sido ampliamente estudiada. Para buscar una consulta en un conjunto de datos de canciones, podemos aplicar cualquiera de los algoritmos de reconocimiento de patrones

conocidos. El principal problema es la eficiencia cuando se utilizan grandes conjuntos de datos. Por otro lado, una indexación previa del conjunto de datos, por ejemplo mediante la extracción de motivos, resuelve el problema de la escalabilidad, pero la extracción de motivos usualmente necesita trabajar con repeticiones exactas, haciendo muy imprecisa la construcción de índices robustos a partir de interpretaciones diferentes de mismas canción. Además, al buscar sólo en motivos, la música no presente en los motivos queda oculta. Nuestra propuesta es capaz de resolver intrínsecamente estos problemas. La propia estructura gramatical probabilística codifica tanto los motivos como las variaciones de la melodía, dando más peso a los temas más repetidos, sin necesidad de ser exactos, y posibilitando el aprendizaje a partir de representaciones diferentes de la misma canción, dejando a un lado la necesidad de codificar motivos. Además, no hay diferencia entre buscar una canción entera o buscar una pequeña consulta en el conjunto de datos. Los resultados superan los obtenidos previamente utilizando autómatas probabilísticos de árboles para el mismo corpus. Según los resultados, podemos decir que la clasificación se mejora dividiendo la melodía en compases. Por atro lado, los resultados también muestran un buen rendimiento teniendo en cuenta la totalidad de la melodía, lo cual es importante ya que no siempre está disponible la información de compases.

Por lo tanto, parece que las gramáticas de árbol son capaces de clasificar consultas de fragmentos con imprecisiones (ruido). Aunque la tasa de acierto parece aumentar con la longitud de la consulta es necesario encontrar un equilibrio entre el la tasa de acierto deseada y el tiempo de ejecución. Por otra parte, la clasificación podría utilizarse como una poda para descartar el análisis de toda la melodía en cada una de las gramáticas. En otras palabras, si queremos analizar una melodía larga, podríamos clasificarla usando sólo el principio de esta. A continuación, calculamos la probabilidad de toda la melodía sólo para las mejores clases. De esta manera, podemos mejorar el tiempo de cálculo cuando la longitud de la melodía es grande.

Sistema generativo Finalmente, nuestra última contribución se deriva de la inferencia de las gramáticas de árbol mencionadas. Estas gramáticas pueden ser utilizar para clasificar las melodías como se mostró anteriormente, pero también se podrían utilizar para generar nuevas melodías usando la estructura de los árboles capturados en las reglas gramáticales inferidas a partir de los datos de arboles. De este modo, podríamos construir un sistema que nos permitiese generar nuevas melodías a partir de las gramáticas inferidas. Este sistema podría utilizarse de varias formas. En primer lugar, la posibilidad de generar música a partir de varias bases de datos

como género, estilo o autor. De esta manera, la nueva música tendría la esencia del conjunto de datos capturado en la gramática. En segundo lugar, el mencionado sistema podría utilizarse para ayudar a los sistemas de composición musical a sugerir nuevas ideas, derivadas de los diversos conjuntos de entrenamiento, para escribir nuevas canciones. En tercer lugar, este motor de generación podría ser utilizado en situaciones que requieren obtener más muestras de forma automática. Por ejemplo, en bases de datos que tienen muestras limitadas y no pueden utilizarse con otros metodos de reconocimiento de patrones porque requieren un gran número de muestras. Finalmente, en cualquier situación que requiera la generación automática de nuevas muestras.

Desafortunadamente, la construcción de un sistema generativo requiere una investigación más profunda que está fuera del alcance de esta tesis. Sin embargo, mostramos algunas líneas a seguir y una investigación inicial con algunas ideas a continuación.

Como se señaló anteriormente, las gramáticas de árboles descritas en la tesis pueden utilizarse para clasificar las melodías como se mostró anteriormente, pero también podrían utilizarse para generar nuevas melodías utilizando la estructura de los árboles capturados en las diferentes reglas gramaticales Inferidas a partir de los datos del árbol. De esta manera, podríamos construir un sistema que nos permitiera generar nuevas melodías a partir de las gramáticas inferidas. Este sistema puede construirse utilizando una aproximación aleatoria para generar nuevas melodías a partir de las probabilidades capturadas en las reglas gramaticales. En otras palabras, se genera una nueva melodía usando la formalización del paseo aleatorio (random walk). A partir del símbolo de inicial, una nueva regla se selecciona aleatoriamente en cada paso. El proceso termina cuando todos los símbolos actuales son terminales. En este punto, se obtiene un nuevo árbol (melodía). Después de haber hecho algunas pruebas preliminares, algunos problemas han aparecido. Uno de ellos es que los árboles generados pueden ser muy profundos y tienen pocos compases. Una rápida solución para resolver este problema puede ser seleccionar los árboles según la profundidad y el número de compases. Otra solución puede ser cambiar el tempo de acuerdo a la profundidad del árbol. Por otro lado, otra forma de generar las nuevas melodías podría ser generar cadenas de de carácteres con un algoritmo genético y luego analizar esta cadena con las gramáticas de árboles recuperando el árbol de análisis de mayor probabilidad de esta cadena de alturas (tonos).

Por lo tanto, de esta forma se obtiene una nueva melodía con las alturas (tonos) generados a partir del algoritmo genético y la estructura árbol obtenida a partir de las gramáticas de árboles.

En cuanto al problema de cómo evaluar las melodías generadas, una posible idea es usar una similitud de edición aprendida con el enfoque descrito en esta tesis, el cual es totalmente independiente de las gramáticas inferidas. De esta manera, podríamos tener un indicador para evaluar las melodías generadas.

Finalmente, después de observar los resultados preliminares de las melodías generadas parece que los modelos están orientados a la compases. Este hecho también se percibió en los autómatas de los árboles que se resolvió utilizando el modelo de tiempo dinámico descrito en la sección 5.3.2. Sin embargo, parece adecuado añadir más información sobre toda la estructura. Por lo tanto, este hecho abre la puerta para estudiar la incorporación de información armónica a la estructura de árbol. Una posible solución es generar una secuencia armónica y luego generar cada compás del árbol de la secuencia hasta que toda la melodía se genera completamente. Desafortunadamente, esta es otra historia y no sabemos si se resolverá en un breve lapso de tiempo.

C.6 Publicaciones

Algunas partes de esta tesis han sido publicadas en revistas y actas de conferencias. A continuacióon mostramos una lista de los artículos en orden cronológico:

- Bellet, A; Bernabeu, J.F.; Habrard, A.; Sebban, M
  "Learning discriminative tree edit similarities for linear classification - Application to melody recognition"
  Neurocomputing, vol. 214, pp. 155-161 (2016)

- Bernabeu, J. F.; Calera-Rubio, J.; Iñesta, J. M.; Rizo, D.
  "Query Parsing Using Probabilistic Tree Grammars"
  5th workshop on Music and Machine Learning, MML 2012, Edinburgh (2012)

- Bernabeu, J.F.; Calera-Rubio, J.; Iñesta, J.M.
  "Classifying melodies using tree grammars"
  Pattern Recognition and Image Analysis: 5th Iberian Conference, IbPRIA 2011, Las Palmas de Gran Canaria, Spain, June 8-10, 2011.
  Lecture Notes in Computer Science, vol. 6669, pp. 572–579 (2011)

- Bernabeu, Jose F. and Calera-Rubio, Jorge and Iñesta, José M. and Rizo, David

"Melodic Identification Using Probabilistic Tree Automata"
Journal of New Music Research, vol. 40, pp. 93-103 (2011)

- Bernabeu, J.F., Calera-Rubio, J., Iñesta, J.M., Rizo, D.
"Tree language automata for melody recognition"
Actas del II Workshop de Reconocimiento de Formas y Análisis de Imágenes (AERFAI), ISBN: 978-84-92812-66-0, pp. 17-22, Valencia, Spain (2010)

- Bernabeu, J.F., Calera-Rubio, J., Iñesta, J.M., Rizo, D.
"A probabilistic approach to melodic similarity"
Proceedings of MML 2009, pp. 48-53 (2009)

Bibliography

Pieter Adriaans and Ceriel Jacobs. *Using MDL for Grammar Induction*, pages 293–306. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-45265-2. doi: 10.1007/11872436_24. URL http://dx.doi.org/10.1007/11872436_24. (Cited on page 13).

Pieter Adriaans, Marten Trautwein, and Marco Vervoort. *Towards High Speed Grammar Induction on Large Text Corpora*, pages 173–186. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000. ISBN 978-3-540-44411-4. doi: 10.1007/3-540-44411-4_11. URL http://dx.doi.org/10.1007/3-540-44411-4_11. (Cited on page 13).

Carlos Agon, Gérard Assayag, J. Fineberg, and Camilo Rueda. Kant: A critique of pure quantification. In *Proc. of the 1994 International Computer Music Conference*, pages 52–59, Aarhus, Denmark, 1994. (Cited on pages 7 and 33).

Noga Alon, Shai Ben-David, Nicolò Cesa-Bianchi, and David Haussler. Scale-sensitive dimensions, uniform convergence, and learnability. *J. ACM*, 44(4): 615–631, July 1997. ISSN 0004-5411. doi: 10.1145/263867.263927. URL http://doi.acm.org/10.1145/263867.263927. (Cited on page 23).

Greg Aloupis, Thomas Fevens, Stefan Langerman, Tomomi Matsui, Antonio Mesa, Yurai Nuñez, David Rappaport, and Godfried T. Toussaint. Algorithms for computing geometric measures of melodic similarity. *Computer Music Journal*, 30(3):67–76, 2006. (Cited on page 5).

Gérard Assayag, Camilo Rueda, Mikael Laurson, Carlos Agon, and Olivier Delerue. Computer-assisted composition at ircam: From patchwork to openmusic. *Computer Music Journal*, 23(3):59–72, 1999. (Cited on pages 6 and 31).

J.-J. Aucouturier and Pachet F. Improving timbre similarity: How high is the sky? *Journal of Negative Results in Speech and Audio Sciences*, 1(1), 2004. (Cited on page 2).

Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. ISBN 020139829X. (Cited on page 25).

Mira Balaban. The music structures approach to knowledge representation for music processing. *Computer Music Journal*, 20(2):96–111, 1996. (Cited on pages 6 and 31).

M.-F. Balcan and A. Blum. On a Theory of Learning with Similarity Functions. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pages 73–80, 2006. (Cited on pages 45, 49, and 50).

M.-F. Balcan, A. Blum, and N. Srebro. Improved Guarantees for Learning via Similarity Functions. In *Proceedings of the 21st Annual Conference on Learning Theory (COLT)*, pages 287–298, 2008. (Cited on pages 11, 45, 46, 47, 48, 49, 50, and 52).

Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3: 463–482, 2002. URL http://www.jmlr.org/papers/v3/bartlett02a.html. (Cited on page 23).

Aurélien Bellet, Amaury Habrard, and Marc Sebban. Good edit similarity learning by loss minimization. *Machine Learning*, 89(1-2):5–35, 2012. (Cited on pages 7, 11, 48, 49, 54, and 110).

Aurélien Bellet, Amaury Habrard, and Marc Sebban. *Metric Learning*. Morgan & Claypool Publishers, 2015. (Cited on page 40).

Shai Ben-David, Nadav Eiron, and Philip M. Long. On the difficulty of approximately maximizing agreements. *Journal of Computer and System Sciences*, 66(3):496 – 514, 2003. ISSN 0022-0000. doi: https://doi.org/10.1016/S0022-0000(03)00038-2. URL http://www.sciencedirect.com/science/article/pii/S0022000003000382. (Cited on page 21).

Shai Ben-David, David Loker, Nathan Srebro, and Karthik Sridharan. Minimizing the misclassification error rate using a surrogate convex loss. In *ICML*. icml.cc / Omnipress, 2012. URL http://dblp.uni-trier.de/db/conf/icml/icml2012.html#Ben-DavidLSS12. (Cited on page 21).

M. Bernard, A. Habrard, and M. Sebban. Learning stochastic tree edit distance. In *Proceedings of the 17th European Conference on Machine Learning (ECML)*, volume 4212 of *LNCS*, pages 42–53. Springer, 2006. (Cited on pages 42 and 43).

M. Bernard, L. Boyer, A. Habrard, and M. Sebban. Learning probabilistic models of tree edit distance. *Pattern Recognition*, 41(8):2611–2629, 2008. (Cited on pages 10, 42, 43, and 55).

Enrico Bertini, Andrada Tatu, and Daniel A. Keim. Quality Metrics in High-Dimensional Data Visualization: An Overview and Systematization. *IEEE Symposium on Information Visualization (InfoVis)*, 17(12):pages 2203–2212, December 2011. (Cited on page 25).

M. Bilenko and R. J. Mooney. Adaptive Duplicate Detection Using Learnable String Similarity Measures. In *Proceeding of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 39–48, 2003. (Cited on page 41).

P. Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1-3):217–239, 2005. (Cited on pages 9, 28, 44, and 45).

Steven G. Blackburn. *Content Based Retrieval and Navigation of Music Using Melodic Pitch Contours*. PhD thesis, University of Southampton, Southampton, UK, September 2000. (Cited on pages 6 and 32).

Rens Bod. A general parsing model for music and language. *Music and Artificial Intelligence*, pages 77–90, 2002. (Cited on pages 5 and 31).

S. Boucheron, G. Lugosi, and O. Bousquet. *Concentration Inequalities*, volume Lecture Notes in Artificial Intelligence 3176, pages 208–240. Springer, Heidelberg, Germany, 2004. (Cited on page 22).

O. Bousquet and A. Elisseeff. Algorithmic stability and generalization performance. In *Advances in Neural Information Processing Systems 13*, pages 196–202, Cambridge, MA, USA, April 2001. Max-Planck-Gesellschaft, MIT Press. (Cited on page 23).

O. Bousquet and A. Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, 2:499–526, 2002. (Cited on page 23).

Olivier Bousquet, Stphane Boucheron, and Gbor Lugosi. Introduction to statistical learning theory. In Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rtsch, editors, *Advanced Lectures on Machine Learning*, volume 3176 of *Lecture Notes in Computer Science*, pages 169–207. Springer, 2003. ISBN 3-540-23122-6. (Cited on page 17).

L. Boyer, Y. Esposito, A. Habrard, J. Oncina, and M. Sebban. Sedil: Software for edit distance learning. In *Proceeding of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD), Part II*, volume 5212 of *LNCS*, pages 672–677, 2008. available from `http://labh-curien.univ-st-etienne.fr/SEDiL/`. (Cited on page 55).

Laurent Boyer, Amaury Habrard, and Marc Sebban. Learning metrics between tree structured data: Application to image recognition. In *ECML*, pages 54–66, 2007. (Cited on pages 10, 42, and 43).

Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jenifer C. Lai, and Robert L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18:18–4, 1990. (Cited on page 62).

Jorge Calera-rubio and Rafael C. Carrasco. Computing the relative entropy between regular tree languages, 1998. (Cited on pages 68 and 69).

Pedro Latorre Carmona, J. Salvador Sánchez, and Ana L. N. Fred, editors. *ICPRAM 2012 - Proceedings of the 1st International Conference on Pattern Recognition Applications and Methods, Volume 1, Vilamoura, Algarve, Portugal, 6-8 February, 2012*, 2012. SciTePress. ISBN 978-989-8425-98-0. (Cited on page 140).

Ali Taylan Cemgil, Bert Kappen, and Peter Desain. Rhythm quantization for transcription. *Computer Music Journal*, 24(2):60–76, 2000. (Cited on pages 7 and 33).

Eugene Charniak. Tree-bank grammars. In *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1031–1036, 1996. (Cited on page 97).

Chaudhuri. Solution of an open problem on probabilistic grammars. *IEEE Trans. Comput.*, 32(8):748–750, 1983. ISSN 0018-9340. (Cited on page 65).

Alexander Clark. *Unsupervised Language Acquisition: Theory and Practice*. PhD thesis, COGS, University of Sussex, 2001. URL papers/thesis.pdf. (Cited on page 13).

Michael Collins and Nigel Duffy. Convolution kernels for natural language. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14 — Proceedings of the 2001 Neural Information Processing Systems Conference (NIPS 2001), December 3-8, 2001, Vancouver, British Columbia, Canada*, pages 625–632. MIT Press, Cambridge, MA, USA, 2002. URL http://books.nips.cc/papers/files/nips14/AA58.pdf. (Cited on page 30).

C. Cortes, P. Haffner, and M. Mohri. Rational Kernels: Theory and Algorithms. *Journal of Machine Learning Research (JMLR)*, 5:1035–1062, 2004. (Cited on page 29).

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20 (3):273–297, September 1995. ISSN 0885-6125. doi: 10.1023/A:1022627411411. URL http://dx.doi.org/10.1023/A:1022627411411. (Cited on pages 21 and 25).

François Coste and Jacques Nicolas. *Inference of finite automata: Reducing the search space with an ordering of pairs of states*, pages 37–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-69781-7. doi: 10.1007/BFb0026669. URL http://dx.doi.org/10.1007/BFb0026669. (Cited on page 13).

T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, September 2006. ISSN 0018-9448. doi: 10.1109/TIT.

1967.1053964. URL http://dx.doi.org/10.1109/TIT.1967.1053964. (Cited on page 24).

N.N. Dalvi, P. Bohannon, and F. Sha. Robust web extraction: an approach based on a probabilistic tree-edit model. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 335–348, 2009. (Cited on pages 10 and 43).

M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. *Atlas of protein sequence and structure*, 5(suppl 3):345–351, 1978. (Cited on pages 27 and 40).

Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, New York, NY, USA, 2010. ISBN 0521763169, 9780521763165. (Cited on page 13).

Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977. (Cited on page 41).

F. Denis, E. Gilbert, A. Habrard, F. Ouardi, and M. Tommasi. Relevant representations for the inference of rational stochastic tree languages. In *Proceedings of the 9th International Colloquium on Grammatical Inference (ICGI)*, volume 5278 of *LNCS*, pages 57–70. Springer, 2008. (Cited on page 54).

Shyamala Doraisamy. *Polyphonic Music Retrieval: The N-gram approach*. PhD thesis, Imperial College London, London, UK, 2004. (Cited on page 5).

J. Stephen Downie. *Evaluating a Simple Approach to Music Information Retrieval: Conceiving Melodic N-Grams as Text*. PhD thesis, University of Western Ontario, Canada, July 1999. (Cited on pages 3, 5, and 62).

Frank Drewes and Johanna Högberg. An algebra for tree-based music generation. In *CAI*, pages 172–188, 2007. (Cited on pages 6 and 32).

Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons Inc, 1973. ISBN 0471223611. (Cited on page 70).

Douglas Eck and Norman Casagrande. Finding meter in music using an autocorrelation phase matrix and shannon entropy. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 504–509, 2005. (Cited on pages 6 and 33).

Charles Elkan. Using the triangle inequality to accelerate k-means. In Tom Fawcett and Nina Mishra, editors, *ICML*, pages 147–153. AAAI Press, 2003.

ISBN 1-57735-189-4. URL http://dblp.uni-trier.de/db/conf/icml/icml2003.html#Elkan03. (Cited on page 26).

Martin Emms. On stochastic tree distances and their training via expectation-maximisation. In *Proceedings of the 1st International Conference on Pattern Recognition Applications and Methods - ICPRAM 2012 - Volume 1*, pages 144–153, 2012. (Cited on pages 10 and 43).

Martin Emms and Hector-Hugo Franco-Penya. On order equivalences between distance and similarity measures on sequences and trees. In Carmona et al. (2012), pages 15–24. ISBN 978-989-8425-98-0. (Cited on page 28).

H. Freeman. Computer Processing of Line-Drawing Images. *ACM Computing Surveys*, 6:57–97, 1974. (Cited on page 41).

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, EuroCOLT '95, pages 23–37, London, UK, UK, 1995. Springer-Verlag. ISBN 3-540-59119-2. URL http://dl.acm.org/citation.cfm?id=646943.712093. (Cited on page 21).

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *Ann. Statist.*, 28(2):337–407, 04 2000. doi: 10.1214/aos/1016218223. URL http://dx.doi.org/10.1214/aos/1016218223. (Cited on page 21).

William Gale. Good-turing smoothing without tears. *Journal of Quantitative Linguistics*, 2, 1994. (Cited on page 76).

X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *Pattern Analysis & Applications*, 13(1):113–129, 2010. (Cited on page 29).

P. García and E. Vidal. Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):920–925, 1990. ISSN 0162-8828. doi: http://doi.ieeecomputersociety.org/10.1109/34.57687. (Cited on pages 62 and 65).

Pedro García. Learning k-testable tree sets from positive data. Technical Report DSIC/II/46/1993, Universidad Politecnica de Valencia, 1993. (Cited on pages 62, 66, 83, and 111).

E. Gilbert and D. Conklin. A probabilistic context-free grammar for melodic reduction. *International Workshop on Artificial Intelligence and Music at IJCAI-07. Twentieth International Joint Conference on Artificial Intelligence, Hyderabad, India*, 2007. (Cited on pages 5, 31, and 35).

Maarten Grachten, Josep Lluís Arcos, and Ramon López De Mántaras. Melody retrieval using the implication/realization model. In *6th Inter. Conf. On Music information retrieval. ISMIR 2005. (First prize of the MIREX Symbolic Melodic Similarity Contest).*, 2005. (Cited on page 5).

Susan L. Graham, , and Michael Harrison Walter L. Ruzzo. An improved context-free recognizer. *ACM Trans. Program. Lang. Syst.*, 2(3):415–462, July 1980. ISSN 0164-0925. doi: 10.1145/357103.357112. URL http://doi.acm.org/10.1145/357103.357112. (Cited on page 91).

A. Habrard, José M. Iñesta, David Rizo, and M. Sebban. Melody recognition with learned edit distances. *LNCS*, 5342:86–96, 2008. (Cited on pages 6, 33, 53, 56, 80, and 109).

S. Handel. *Listening: An Introduction to the Perception of Auditory Events.* A Bradford book. MIT Press, 1989. ISBN 9780262081795. URL https://books.google.es/books?id=LChBcAAACAAJ. (Cited on page 4).

Pierre Hanna, Matthias Robine, Pascal Ferraro, and Julien Allali. Improvements of alignment algorithms for polyphonic music retrieval. In *Computer Music Modeling and Retrieval 2008*, Copenhague, 2008. (Cited on pages 6 and 33).

M. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1978. ISBN 0201029553. (Cited on page 90).

David Haussler. Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, University of California at Santa Cruz, Santa Cruz, CA, USA, 1999. URL http://citeseer.ist.psu.edu/haussler99convolution.html. (Cited on page 30).

S. Henikoff and Henikoff. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci U S A J*, 89:10915–9, November 1992. (Cited on pages 27 and 40).

Marijn J. H. Heule and Sicco Verwer. *Exact DFA Identification Using SAT Solvers*, pages 66–79. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-15488-1. doi: 10.1007/978-3-642-15488-1_7. URL http://dx.doi.org/10.1007/978-3-642-15488-1_7. (Cited on page 13).

Johanna Högberg. Wind in the willows - generating music by means of tree transducers. In *CIAA*, pages 153–162, 2005. (Cited on pages 6 and 31).

Jia-Lien Hsu, Arbee L. P. Chen, and C.-C. Liu. Efficient repeating pattern finding in music databases. In *Proceedings of the Seventh International Conference on Information and Knowledge Management*, CIKM '98, pages 281–288, New York,

NY, USA, 1998. ACM. ISBN 1-58113-061-9. doi: 10.1145/288627.288668. URL http://doi.acm.org/10.1145/288627.288668. (Cited on page 3).

Plácido R. Illescas, David Rizo, and José M. Iñesta. Harmonic, melodic, and functional automatic analysis. In *Proc. of the 2007 International Computer Music Conference*, volume I, pages 165–168, 2007. (Cited on page 34).

Tommi Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *In Advances in Neural Information Processing Systems 11*, pages 487–493. MIT Press, 1998. (Cited on page 30).

F. Jelinek. *Statistical Methods for Speech Recognition*. The MIT Press, January 1998. ISBN 0262100665. (Cited on page 62).

Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 321–328. AAAI Press, 2003. (Cited on page 30).

Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, pages 400–401, 1987. (Cited on page 75).

Phillip B. Kirlin and Paul E. Utgoff. A framework for automated schenkerian analysis. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR 2008)*, September 2008. (Cited on pages 6 and 31).

P.N. Klein. Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th European Symposium on Algorithms (ESA)*, volume 1461 of *LNCS*, pages 91–102. Springer, 1998. (Cited on page 44).

T. Knuutila. Inference of k-testable tree languages. In *Bunke (Ed.), Advances in Structural and Syntactic Pattern Recognition (Proc. of the S+SSPR'92), World Scientific, Singapore*, 1993. (Cited on pages 62, 65, 66, and 85).

V. Koltchinskii. Rademacher penalties and structural risk minimization. *IEEE Transactions on Information Theory*, 47(5):1902–1914, Jul 2001. ISSN 0018-9448. doi: 10.1109/18.930926. (Cited on page 23).

Raymond Kosalaa, Hendrik Blockeela, Maurice Bruynooghea, and Jan Van den Bussche. Information extraction from structured documents using k-testable tree automaton inference. *Data & Knowledge Engineering*, 58(2):129–158, August 2006. (Cited on pages 13, 61, and 85).

Carol L. Krumhansl. The psychological representation of musical pitch in a tonal context. *Cognitive Psychology*, 11(3):346 – 374, 1979. ISSN 0010-0285.

doi: https://doi.org/10.1016/0010-0285(79)90016-1. URL http://www.sciencedirect.com/science/article/pii/0010028579900161. (Cited on page 3).

Jim Z. C. Lai, Yi-Ching Liaw, and Julie Liu. Fast k-nearest-neighbor search based on projection and triangular inequality. *Pattern Recognition*, 40(2):351–359, 2007. doi: 10.1016/j.patcog.2006.04.024. URL https://doi.org/10.1016/j.patcog.2006.04.024. (Cited on page 26).

Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. *Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm*, pages 1–12. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-68707-8. doi: 10.1007/BFb0054059. URL http://dx.doi.org/10.1007/BFb0054059. (Cited on page 13).

John Langford. Tutorial on practical prediction theory for classification. *J. Mach. Learn. Res.*, 6:273–306, December 2005. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1046920.1058111. (Cited on page 22).

C. S. Lee. The rhythmic interpretation of simple musical sequences: Towards a perceptual model. In R. West, P. Howell, and I. Cross, editors, *Musical Structure and Cognition*, pages 53–69. Academic Press, London, 1985. (Cited on pages 5 and 31).

Kjell Lemström. *String Matching Techniques for Music Retrieval*. PhD thesis, University of Helsinki, Finland, November 2000. (Cited on page 5).

F. Lerdahl and R. Jackendoff. *A Generative Theory of Tonal Music*. MIT Press, Cambridge, Massachusetts, 1983. (Cited on pages 6, 31, and 35).

C Leslie, E Eskin, and W S Noble. The spectrum kernel: a string kernel for svm protein classification. *Pac Symp Biocomput*, pages 564–575, 2002. URL http://www.ncbi.nlm.nih.gov/pubmed/11928508. (Cited on page 29).

C. Leslie, E. Eskin, J. Weston, and WS. Noble. Mismatch string kernels for svm protein classification. In *Advances in Neural Information Processing Systems 15*, pages 1417–1424, Cambridge, MA, USA, October 2003. Max-Planck-Gesellschaft, MIT Press. (Cited on page 29).

V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics-Doklandy*, 6:707–710, 1966. (Cited on pages 11 and 27).

Daniel J. Levitin. Music, cognition, and computerized sound. chapter Memory for Musical Attributes, pages 209–227. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-03256-2. URL http://dl.acm.org/citation.cfm?id=304632.304666. (Cited on page 3).

H. Li and T. Jiang. A class of edit kernels for SVMs to predict translation initiation sites in eukaryotic mRNAs. In *Proceedings of the 8th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 262–271, 2004. (Cited on page 29).

Thomas Lidy and Andreas Rauber. Evaluation of feature extractors and psycho-acoustic transformations for music genre classification. In *ISMIR 2005, 6th International Conference on Music Information Retrieval, London, UK, 11-15 September 2005, Proceedings*, pages 34–41, 2005. URL http://ismir2005.ismir.net/proceedings/1033.pdf. (Cited on page 2).

Pertusa A. Iñesta J.M. Lidy T., Rauber A. Improving genre classification by combination of audio and symbolic descriptors using a transcription system. In *Proc. of the 8th Int. Conf. on Music Information Retrieval, ISMIR 2007*, pages 61–66, Vienna, Austria, 2007. (Cited on page 2).

S. Lloyd. Least squares quantization in pcm. *IEEE Trans. Inf. Theor.*, 28(2):129–137, September 2006. ISSN 0018-9448. doi: 10.1109/TIT.1982.1056489. URL http://dx.doi.org/10.1109/TIT.1982.1056489. (Cited on page 25).

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *J. Mach. Learn. Res.*, 2:419–444, March 2002. ISSN 1532-4435. doi: 10.1162/153244302760200687. URL http://dx.doi.org/10.1162/153244302760200687. (Cited on page 29).

Alan Marsden. Representing melodic patterns as networks of elaborations. *Computers and the Humanities*, 35(1):37–54, 2001. (Cited on pages 6 and 31).

Alan Marsden. Generative structural representation of tonal music. *Journal of New Music Research*, 34(4):409 – 428, December 2005. (Cited on pages 6, 31, and 35).

Alan Marsden. Automatic derivation of musical structure: A tool for research on schenkerian analysis. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR'07)*, Vienna, Austria, September 2007. (Cited on pages 6 and 31).

Andrew McCallum, Kedar Bellare, and Fernando Pereira. A Conditional Random Field for Discriminatively-trained Finite-state String Edit Distance. In *Proceedings of 21st Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 388–395, 2005. (Cited on page 41).

Yashar Mehdad. Automatic cost estimation for tree edit distance using particle swarm optimization. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint*

Conference on Natural Language Processing of the AFNLP (ACL-IJCNLP), pages 289–292, 2009. (Cited on page 10).

Benoît Meudic. Automatic meter extraction from midi files. In *Journées d'informatique musicale. Marseille: Centre National de Création Musicale*, 2002. (Cited on pages 6 and 33).

L. Micó and J. Oncina. Comparison of fast nearest neighbour classifiers for handwritten character recognition. *Pattern Recognition Letters*, 19:351–356, 1998. (Cited on pages 27 and 40).

María Luisa Micó, José Oncina, and Enrique Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. *Pattern Recogn. Lett.*, 15(1): 9–17, January 1994. ISSN 0167-8655. doi: 10.1016/0167-8655(94)90095-7. URL http://dx.doi.org/10.1016/0167-8655(94)90095-7. (Cited on page 26).

Mehryar Mohri and Afshin Rostamizadeh. Stability bounds for non-i.i.d. processes. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 1025–1032, 2007. URL http://papers.nips.cc/paper/3239-stability-bounds-for-non-iid-processes. (Cited on page 24).

Mehryar Mohri and Afshin Rostamizadeh. Stability bounds for stationary phi-mixing and beta-mixing processes. *Journal of Machine Learning Research*, 11: 789–814, 2010. doi: 10.1145/1756006.1756032. URL http://doi.acm.org/10.1145/1756006.1756032. (Cited on page 24).

Marcel Mongeau and David Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24(3):161–175, 1990. (Cited on pages 5, 6, and 33).

David W. Mount. *Bioinformatics: sequence and genome analysis*, chapter Phylogenetic Prediction. CSHL press, 2004. (Cited on page 27).

Katsuhiko Nakamura and Takashi Ishiwata. *Synthesizing Context Free Grammars from Sample Strings Based on Inductive CYK Algorithm*, pages 186–195. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000. ISBN 978-3-540-45257-7. doi: 10.1007/978-3-540-45257-7_15. URL http://dx.doi.org/10.1007/978-3-540-45257-7_15. (Cited on page 13).

Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443 – 453, 1970.

ISSN 0022-2836. doi: DOI:10.1016/0022-2836(70)90057-4. URL http://www.sciencedirect.com/science/article/B6WK7-4DN8W3K-7X/2/0d99b8007b44cca2d08a031a445276e1. (Cited on page 27).

M. Neuhaus and H. Bunke. A probabilistic approach to learning costs for graph edit distance. In *Proceedings of the 17th International Conference on Pattern Recognition (ICPR)*, pages 389–393. IEEE, 2004. (Cited on page 10).

M. Neuhaus and H. Bunke. Edit distance-based kernel functions for structural pattern classification. *Pattern Recognition*, 39:1852–1863, 2006. (Cited on page 29).

Michel Neuhaus and Horst Bunke. Automatic learning of cost functions for graph edit distance. *Inf. Sci.*, 177(1):239–247, 2007. doi: 10.1016/j.ins.2006.02.013. URL http://dx.doi.org/10.1016/j.ins.2006.02.013. (Cited on page 43).

Hermann Ney, Ute Essen, and Reinhard Kneser. On the estimation of small probabilities by leaving-one-out. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17 (12):1202–1212, 1995. (Cited on pages 62, 67, 70, 71, and 98).

J. Oncina and M. Sebban. Learning Stochastic Edit Distance: application in handwritten character recognition. *Pattern Recognition*, 39(9):1575–1587, 2006. (Cited on pages 41 and 42).

Nicola Orio. Music retrieval: A tutorial and review. *Found. Trends Inf. Retr.*, 1(1):1–96, January 2006. ISSN 1554-0669. doi: 10.1561/1500000002. URL http://dx.doi.org/10.1561/1500000002. (Cited on page 1).

Jean-Franois Paiement. *Probabilistic Models for Music*. PhD thesis, Université de Montréal, 2008. (Cited on page 3).

Bryan Pardo and Manan Sanghi. Polyphonic musical sequence alignment for database search. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 215–222, 2005. (Cited on pages 6 and 33).

Mateusz Pawlik and Nikolaus Augsten. Rted: A robust algorithm for the tree edit distance. *Proc. VLDB Endow.*, 5(4):334–345, December 2011. ISSN 2150-8097. doi: 10.14778/2095686.2095692. URL http://dx.doi.org/10.14778/2095686.2095692. (Cited on page 28).

C. Pérez-Sancho. *Stochastic Language Models for Music Information Retrieval*. PhD thesis, Alicante, Spain, July 2009. (Cited on page 113).

Georgios Petasis, Georgios Paliouras, Constantine D. Spyropoulos, and Constantine Halatsis. *eg-GRIDS: Context-Free Grammatical Inference*

from Positive Examples Using Genetic Search, pages 223–234. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-30195-0. doi: 10.1007/978-3-540-30195-0_20. URL http://dx.doi.org/10.1007/978-3-540-30195-0_20. (Cited on page 13).

Alberto Pinto and Paolo Tagliolato. A generalized graph-spectral approach to melodic modeling and retrieval. In *MIR '08: Proceeding of the 1st ACM international conference on Multimedia information retrieval*, pages 89–96, New York, NY, USA, 2008. ACM. (Cited on page 5).

L. R. Rabiner and R. W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall, Englewood Cliffs, NJ, 1978. (Cited on page 3).

E. Rich. *Automata, Computability and Complexity: Theory and Applications*. Pearson Prentice Hall, 2008. ISBN 9780132288064. URL https://books.google.es/books?id=lIuu53IcKWoC. (Cited on page 91).

J. R. Rico-Juan, J. Calera-Rubio, and R. C. Carrasco. Smoothing and compression with stochastic k-testable tree languages. *Pattern Recognition*, 38(9):1420–1430, 2005. (Cited on pages 13, 14, 61, 73, 75, and 85).

E. S. Ristad and P. N. Yianilos. Learning String-Edit Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532, 1998. (Cited on pages 41 and 42).

D. Rizo and J.M. Iñesta. *Tree-structured representation of melodies for comparison and retrieval*, chapter -, page 155. -, 2002. (Cited on page 32).

D. Rizo, K. Lemstrm, and J. M. Iñesta. Tree representation in combined polyphonic music comparison. *Computer Music Modeling and Retrieval. Genesis of Meaning in Sound and Music. Lecture Notes in Computer Science*, 5493:177–195, 2009. ISSN 978-3-642-02517-4. (Cited on pages 33 and 53).

David Rizo. *Symbolic music comparison with tree data structures*. PhD thesis, Universidad de Alicante, November 2010. (Cited on pages 6, 32, 53, and 109).

David Rizo, F. Moreno-Seco, and José M. Iñesta. Tree-structured representation of musical information. *LNCS*, 2652:838–846, 2003. (Cited on page 6).

Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. Are loss functions all the same? *Neural Comput.*, 16(5): 1063–1076, May 2004. ISSN 0899-7667. doi: 10.1162/089976604773135104. URL http://dx.doi.org/10.1162/089976604773135104. (Cited on page 21).

H. Saigo, J.-P. Vert, N. Ueda, and T. Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 20(11):1682–1689, 2004. (Cited on pages 29 and 42).

Hiroto Saigo, Jean-Philippe Vert, and Tatsuya Akutsu. Optimizing amino acid substitution matrices with a local alignment kernel. *BMC Bioinformatics*, 7 (246):1–12, 2006. (Cited on page 42).

Yasubumi Sakakibara. Efficient learning of context-free grammars from positive structural examples. *Inf. Comput.*, 97(1):23–60, 1992. ISSN 0890-5401. (Cited on page 65).

Yasubumi Sakakibara, Michael Brown, Richard Hughey, Saira Mian, Kimmen Sjölander, Rebecca C. Underwood, and David Haussler. *Recent methods for RNA modeling using stochastic context-free grammars*, pages 289–306. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994. ISBN 978-3-540-48450-9. doi: 10.1007/3-540-58094-8_25. URL http://dx.doi.org/10.1007/3-540-58094-8_25. (Cited on page 13).

G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, November 1975. ISSN 0001-0782. doi: 10.1145/361219.361220. URL http://doi.acm.org/10.1145/361219.361220. (Cited on page 25).

Joan-Andreu Sánchez and José-Miguel Benedí. Consistency of stochastic context-free grammars from probabilistic estimation based on growth transformations, 1997. (Cited on page 65).

B. Schölkopf, J. Weston, E. Eskin, C. Leslie, and WS. Noble. A kernel approach for learning from almost orthogonal patterns. In *Principles of Data Mining and Knowledge Discovery, Lecture Notes in Computer Science*, volume 2430/2431 of *Lecture Notes in Computer Science*, pages 511–528, Berlin, Germany, 2002. Max-Planck-Gesellschaft, Springer. (Cited on page 29).

Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, MA, USA, 2001. ISBN 0262194759. (Cited on page 25).

Eleanor Selfridge-Field. *Beyond MIDI: The handbook of musical codes.* MIT Press, Cambridge, Massachusetts, USA, 1997. (Cited on page 36).

Eleanor Selfridge-Field. Conceptual and representational issues in melodic comparison. In Walter B. Hewlett and Eleanor Selfridge-Field, editors, *Melodic Similarity: Concepts, Procedures, and Applications. Computing in Musicology*, volume 11, chapter 1, pages 223–230. MIT Press, 1998. (Cited on page 1).

S.M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6 (6):184–186, 1977. (Cited on pages IX, 28, 42, 44, 45, 49, 55, and 80).

Kilho Shin and Tetsuji Kuboyama. A generalization of haussler's convolution kernel: Mapping kernel. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 944–951, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390275. URL http://doi.acm.org/10.1145/1390156.1390275. (Cited on page 30).

Kilho Shin, Marco Cuturi, and Tetsuji Kuboyama. Mapping kernels for trees. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 961–968, 2011. (Cited on page 30).

Josef Sivic and Andrew Zisserman. Efficient visual search of videos cast as text retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(4):591–606, April 2009. ISSN 0162-8828. doi: 10.1109/TPAMI.2008.111. URL http://dx.doi.org/10.1109/TPAMI.2008.111. (Cited on page 25).

Michael Skalak, Jinyu Han, and Bryan Pardo. Speeding melody search with vantage point trees. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR 2008)*, September 2008. (Cited on pages 6 and 32).

Alan Smaill, Geraint A. Wiggins, and Mitch Harris. Hierarchical music representation for composition and analysis. *Computers and the Humanities*, 27(1):7–17, 01 1993. (Cited on pages 6 and 31).

T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. In *Journal of Molecular Biology*, volume 147(1), pages 195–197, 1981. (Cited on page 27).

Stephen W Smoliar. A computer aid for schenkerian analysis. In *ACM 79: Proceedings of the 1979 annual conference*, pages 110–115, New York, NY, USA, 1979. ACM. (Cited on pages 6 and 31).

Andreas Stolcke. *Bayesian learning of probabilistic language models*. PhD thesis, University of California, Berkeley, 1994. (Cited on page 13).

Andreas Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21:165–201, 1995. (Cited on pages 91, 94, and 103).

Andreas Stolcke and Jonathan Segal. Precise n-gram probabilities from stochastic context-free grammars, 1994. (Cited on page 68).

Atsuhiro Takasu. Bayesian Similarity Model Estimation for Approximate Recognized Text Search. In *Proceedings of the 10th International Conference on Document Analysis and Recognition (ICDAR)*, pages 611–615, 2009. (Cited on page 41).

Luke Tanur. *A Geometric Approach to Pattern Matching in Polyphonic Music*. PhD thesis, University of Waterloo, 2005. (Cited on page 5).

Koji Tsuda, Taishin Kin, and Kiyoshi Asai. Marginalized kernels for biological sequences. *Bioinformatics*, 18(1):S268, 2002. doi: 10.1093/bioinformatics/18.suppl_1.S268. URL +http://dx.doi.org/10.1093/bioinformatics/18.suppl_1.S268. (Cited on page 30).

Rainer Typke. *Music Retrieval based on Melodic Similarity*. PhD thesis, Utrecht University, Netherlands, February 2007. (Cited on page 5).

Rainer Typke, Frans Wiering, and Remco C. Veltkamp. A survey of music information retrieval systems. In *In ISMIR*, pages 153–160, 2005. (Cited on page 1).

George Tzanetakis and Perry R. Cook. Musical genre classification of audio signals. *IEEE Trans. Speech and Audio Processing*, 10(5):293–302, 2002. doi: 10.1109/TSA.2002.800560. URL http://dx.doi.org/10.1109/TSA.2002.800560. (Cited on page 2).

George Tzanetakis, Andrey Ermolinskyi, and Perry Cook. Pitch histograms in audio and symbolic music information retrieval. *Journal of New Music Research*, 32(2):143–152, 2003. (Cited on page 2).

Alexandra L. Uitdenbogerd. *Music Information Retrieval Technology*. PhD thesis, RMIT University, Melbourne, Victoria, Australia, July 2002. (Cited on page 5).

Esko Ukkonen, Kjell Lemström, and Veli Mäkinen. Geometric algorithms for transposition invariant content based music retrieval. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2003. (Cited on page 5).

L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984. ISSN 0001-0782. doi: 10.1145/1968.1972. URL http://doi.acm.org/10.1145/1968.1972. (Cited on page 21).

V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971. doi: 10.1137/1116025. URL http://dx.doi.org/10.1137/1116025. (Cited on pages 17 and 22).

Vladimir Vapnik. *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982. ISBN 0387907335. (Cited on page 22).

Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998. (Cited on page 17).

Jarkko Venna, Jaakko Peltonen, Kristian Nybo, Helena Aidos, and Samuel Kaski. Information retrieval perspective to nonlinear dimensionality reduction for data visualization. *J. Mach. Learn. Res.*, 11:451–490, March 2010. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1756006.1756019. (Cited on page 25).

J. L. Verdu-Mas, R. C. Carrasco, and J. Calera-Rubio. Parsing with probabilistic strictly locally testable tree languages. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1040–1050, 2005. (Cited on pages 14, 86, 97, and 98).

Xueyi Wang. A fast exact k-nearest neighbors algorithm for high dimensional search using k-means clustering and triangle inequality. In *IJCNN*, pages 1293–1299. IEEE, 2011. ISBN 978-1-4244-9635-8. URL http://dblp.uni-trier.de/db/conf/ijcnn/ijcnn2011.html#Wang11. (Cited on page 26).

Geraint A. Wiggins, Kjell Lemström, and David Meredith. Sia(m)ese: An algorithm for transposition invariant, polyphonic content-based music retrieval. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2002. (Cited on page 5).

Rui Yang, Panos Kalnis, and Anthony K. H. Tung. Similarity evaluation on tree-structured data. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 754–765, New York, NY, USA, 2005. ACM. ISBN 1-59593-060-4. doi: 10.1145/1066157.1066243. URL http://doi.acm.org/10.1145/1066157.1066243. (Cited on page 28).

Takashi Yokomori. On polynomial-time learnability in the limit of strictly deterministic automata. *Mach. Learn.*, 19(2):153–179, 1995. ISSN 0885-6125. (Cited on page 62).

Menno Matthias Van Zaanen. *Bootstrapping structure into language : alignment-based learning*. PhD thesis, University of Leeds, UK, 2001. URL http://etheses.whiterose.ac.uk/1304/. (Cited on page 13).

Yechezkel Zalcstein. Locally testable languages. *J. Comput. Syst. Sci.*, 6(2):151–167, 1972. (Cited on page 62).

K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, pages 1245–1262, 1989. (Cited on pages 10, 28, 42, 44, 49, and 55).

Philip Zigoris, Damian Eads, and Yi Zhang. Unsupervised learning of tree alignment models for information extraction. In *ICDM Workshops*, pages 45–49, 2006. (Cited on page 10).