Recognition of Pen-based Music Notation with Finite-State Machines

Jorge Calvo-Zaragoza[a,*], Jose Oncina[a]

[a]*Departamento de Lenguajes y Sistemas InformÃąticos, Universidad de Alicante, Carretera San Vicente del Raspeig s/n, 03690 Alicante, Spain*

Abstract

This work presents a data-driven statistical approach with which to recognize pen-based music compositions. Unlike previous works, no assumption is made as regards the handwriting notation, but the system is able to adapt to any kind of style obtained from training data. The series of strokes received as input is mapped onto a stochastic representation, which is combined with a formal language that describes musical symbols in terms of stroke primitives. A Probabilistic Finite-State Automaton is then obtained, which defines probabilities over the set of musical sequences. This model is eventually crossed with a semantic language to avoid sequences that do not make musical sense. Finally, a decoding strategy is applied in order to output a hypothesis concerning the musical sequence actually written. Comprehensive experimentation with several decoding algorithms, stroke similarity measures and probability density estimators are tested and evaluated following different metrics of interest to both user-dependent and user-independent scenarios. The results demonstrate that it is possible to learn the handwriting music style, obtaining competitive performances in each case considered.

Keywords: Pen-based Recognition, Optical Music Recognition, Finite-State Machines

*Corresponding author: Tel.: +349-65-903772; Fax: +349-65-909326

Email addresses: `jcalvo@dlsi.ua.es` (Jorge Calvo-Zaragoza), `oncina@ua.es` (Jose Oncina)

1. Introduction

Despite several efforts to develop light and friendly software for music score edition, many composers still prefer to express their new musical compositions using a pen and paper. Once the artistic process is over, however, they resort to this kind of tools to transcribe the musical content into some kind of machine-readable format. Although this process is not always mandatory, it entails several benefits such as an easier storage, organization, distribution or reproduction of the music scores.

One profitable way in which to perform the whole process is by means of a pen-based music notation recognition system. These systems make use of an electronic pen, with which music symbols are drawn on a digital surface. The system collects user strokes and then processes them to recognize the composition. The goal is to present the score actually written to the user in the desired format. It should be noted that this task can be considered very similar to the Optical Character Recognition (OCR) task, for which pen-based research has been widely carried out (Plamondon & Srihari, 2000; Mondal et al., 2009; Liu et al., 2013). Nevertheless, the complexity of musical notation in comparison to text leads to the need for specific developments (Bainbridge & Bell, 2001).

One straightforward approach that can be used to solve the task stated above is that of resorting to Optical Music Recognition (OMR) systems, whose purpose is to understand music scores from their image. That is, an image can be generated from pen strokes in order to make it pass through a conventional image-based system (offline recognition). Nevertheless, the performance of OMR systems is far from optimal, especially in the case of handwritten notation (Rebelo et al., 2012). Note that the main intention of a pen-based score composition system is to provide musicians with an interface that is as friendly as possible. The musicians should, therefore, be able to compose without having to pay attention to whether or not their handwriting is perfect. However, imperfect handwriting makes it even more difficult than usual to recognize the notation.

Fortunately, pen-based (or online) recognition provides new features that make the task very different to the offline case, some of which include:

- Staff lines: a staff is composed of five parallel lines, on which musical symbols are placed at different heights depending on their pitch. Staff-line detection and removal usually entails an obstacle that most offline OMR systems must overcome (Dalitz et al., 2008), since symbol detection and recognition are based on the accuracy of this step. Nevertheless, in a pen-based system the problem is insignificant because the lines in the staff are handled by the system itself and can be removed effortlessly.

- Segmentation: the input of a pen-based system is naturally segmented by pen strokes. Each stroke is easily detected by pen-down and pen-up actions over the digital surface. This makes it possible to avoid a lot of potential mistakes that may be caused by a bad segmentation in OMR systems.

- Online data: drawing symbols in the pen-based scenario produces a time signal of coordinates indicating the path followed by the pen. Although the image of the score can be rebuilt from the strokes, online data is available to be used during recognition. It is known that this dynamic information is valuable for shape recognition (Kim & Sin, 2014).

All of the above features should lead to the development of specific pen-based algorithms that are able to improve the performance of current offline OMR systems.

This work proposes an approach with which to solve the problems involved in this task by using finite-state machines and dissimilarity measures between strokes. For a given input, the combination of these artifacts is able to produce a probabilistic model that defines the probability of each possible musical sequence. The use of decoding algorithms (for instance, searching for the most probable sequence) provides a hypothesis regarding the sequence actually written.

The use of finite-state machines makes it possible to build a system based on machine learning, signifying that the recognition is adaptable to any kind of handwriting style as long as training data is provided. Furthermore, these models establish an elegant statistical framework to solve the task, which is also supported by the vast background research conducted on finite-state machines (Mohri et al., 2002). It will be shown that this framework enables the easy incorporation of domain information — such as the fact that strokes are single independent units that can be grouped under the same label — and semantic constraints. Unlike that which occurs with similar frameworks such as hidden Markov models, this information prevents the use of pruning methods that may involve a loss of recognition accuracy and representativity (Rabiner, 1989).

The remainder of the paper is structured as follows: Section 2 presents some work related to pen-based music recognition; Section 3 provides details on the construction of the probabilistic model; Section 4 describes the experimentation carried out and the results obtained, and finally, Section 5 shows the main conclusions drawn and discusses some future work.

2. Background

Notwithstanding the benefits provided by pen-based music recognition systems for the composition of music, little attention has been paid to their development.

The first systems for the online recognition of musical scores were based on the use of simple gestures. This is the case of the *Presto* system, developed by Anstice et al. (1996), which received as input short gestures that were generally mnemonic of the actual symbols they represented. These gestures were processed and translated into the actual musical symbols. The system was further improved in a later work (Ng et al., 1998), including new capabilities based on a usability analysis of the previous approach. Poláček et al. (2009) developed a similar idea for use in low-resolution displays. These authors' alphabet consisted of gestures that were similar to conventional music notation, yet restricted to a more simple notation style. The main drawback of these approaches was that they did not provide a natural interface to musicians, who had to learn a new way of writing music.

More recently, many works have dealt with the problem of recognizing pen-based isolated musical symbols. George (2003) used the images generated by the digital pen to learn an Artificial Neural Network with which to recognize symbols. Her experimentation comprised 4 188 music symbols spread over 20 types of musical symbols from 25 different users. Although she completely ignored the time-signal information provided by the e-pen, she was able to obtain an accuracy of around 80 %. Lee et al. (2010b) proposed the use of hidden Markov models for the recognition of some of the most common musical symbols using different features of the shape drawn by the pen. The work was further extended in (Lee et al., 2010a), which considered other recognition schemes such as Naive Bayes or Markov models. These authors' results showed a virtually optimal performance. The conclusions drawn are, unfortunately, difficult to generalize since the experimentation only involved 400 samples, spread over 8 classes of symbols, from a single user. Calvo-Zaragoza & Oncina (2014), meanwhile, presented a free dataset of 15 200 pen-based music symbols written by 100 different musicians, comprising 32 different music symbols. In addition to the dataset, they included an experimental baseline study taking into account several recognition algorithms. Their results reported that the time-signal information provided by the e-pen is able to outperform those classifiers that focus only on the shape drawn.

Nevertheless, while the recognition of isolated symbols might be of interest, the actual focus of this work is on the recognition of pen-based music sequences, which comprises a very different (and actually more difficult) challenge, and far fewer attempts have been made to deal with this issue.

Miyao & Maruyama (2004, 2007) proposed a system based on predefined stroke primitives (such as note-heads, lines, dots, etc.). The combination of the time-series data and image features were employed to recognize strokes using

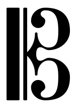
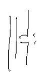
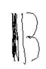
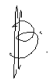
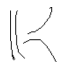
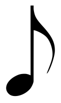
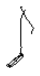
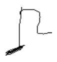
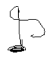
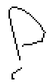
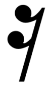
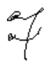
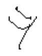
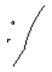
Support Vector Machines. Once the strokes had been classified, musical symbols were reconstructed following a heuristic approach that made use of a set of fixed restrictive rules in order to convert strokes into music symbols. The recognition rates were about 98 and 99 % for strokes and music symbols, respectively. In spite of this performance, the approach did not enable users to write naturally, since the set of construction rules was very restrictive.

Macé et al. (2005) proposed a generic approach for pen-based document recognition applied to music scores. This approach consisted of three modules: a stroke recognizer, a definition of the spatial structure of the document and a pen-based interaction framework. The recognition of the strokes was based on Neural Networks. The spatial structure of the document was modeled by means of *ad-hoc* Context-free Grammars, that took into account chronological and spatial information to define how and where strokes were expected by the system. This meant that any handwriting style that did not fulfill these constraints was not properly recognized. The interactivity framework allowed the user to validate or reject any output of the recognizer. Although no information regarding recognition rates was provided, the system was expected to recognize notes, accidentals, clefs, rests and barlines.

To the best of our knowledge, and supported by relevant reviews such as those by Rebelo et al. (2012) or Fornés & Sánchez (2014), no further research on the pen-based music recognition task has been carried out. As discussed above, the solutions proposed to date are not satisfactory from a user point of view since the only way in which to recognize symbols is by following the rules proposed by each system. These solutions have, therefore, forced users to adapt to the system style when what should be pursued is precisely the opposite.

All of the above reasons signify that there is still a need to develop an appropriate pen-based editing system for music scores. The main goals are to provide an ergonomic interface, which is indeed fulfilled with the use of the e-pen, and to provide an adaptive behavior that will allow musicians to use their own personal handwriting style. It is important to note that this question is not trivial. As an example, Table 1 shows some musical symbols written by different musicians, in which a great variability can be observed. This implies that recognition must be guided by a learning process that will allow the system to know how musicians are going to write their music, instead of resorting to hand-crafted heuristics.

Table 1: Some musical symbols written by different musicians.

Label	Symbol	Musician 1	Musician 2	Musician 3	Musician 4
C-Clef					
Eighth Note					
Sixteenth Rest					

Our proposal follows a stroke-based approach, that is, we consider a finite set of *stroke primitives*. We then use labeled data to learn a language that describes musical symbols obtained from isolated stroke primitives. The strokes received as input are mapped onto the probability of representing each of the primitives considered. In addition, we use a semantic model that describes which musical sequences are formally acceptable.

This knowledge is eventually merged into a finite-state machine (Hopcroft, 1979), since it represents a natural and compact means to express and combine all the information available. Furthermore, these machines represent a well-known established framework, since similar approaches have previously been proposed for speech and text recognition (Mohri et al., 2005).

Once the finite-state machine has been built, a hypothesis about the musical sequence actually written can be provided following a decoding strategy. An in-depth description of how to build such a model is provided in the following section.

3. Pen-Based Music Recognition with Finite-State Machines

This section describes our approach for the recognition of handwritten musical sequences written with an e-pen.

In this work it is assumed that a training set with samples of how isolated musical symbols are written by users is available. This corpus might be obtained by either asking the user to go through a training phase before using the tool or by using an existing dataset (such as that mentioned in the previous section). This will make it possible to define a set of construction rules from stroke primitives to musical symbols.

The probability of each input stroke representing each possible stroke primitive will be computed. This estimation, along with the construction rules defined in the training phase, will be used to obtain a probabilistic machine that is able to give a probability to each of the possible musical sequences. Nevertheless, given the formal system in which music is framed, it is known that there are several sequences of musical symbols that do not make sense. A semantic model will therefore be used to tune this probabilistic machine in order to avoid those sequences that are not well-formed.

Finally, several decoding strategies will be applied to provide a hypothesis that will be considered as a solution to the problem involved in this task.

3.1. Symbol generation from stroke primitives

The input to the system consists of the set of time-ordered strokes drawn by the user. Each symbol may be written with either one single stroke or with several. For instance, a *Half Note* (𝅗𝅥) can be a *white note head* followed by a *stem* (Fig. 1a), or only the *half note* primitive if the symbol is written with a single stroke (Fig. 1b). If symbols are to be recognized from this kind of input, it is necessary to know how each musical symbol can be written from strokes.

(a) Two strokes (b) One stroke

Figure 1: *Half Note* written with different sets of strokes.

Owing to the stroke feature space, it is very unlikely that equal strokes will be written more than once, and using this dataset to learn may therefore be useless. Nevertheless, if *similar* strokes are grouped under the same label, the complexity of this process is reduced, which additionally allows a higher generalization. Since our main intention is to avoid predefined heuristics, the decision as to how to group strokes under the same label must be made according to the training set. This will be explained in greater depth when describing the training stage (Section 3.3).

Once strokes have been assigned a primitive label, musical symbols can be defined from sequences of stroke primitives. Let us denote $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ as the set of musical symbols and $\Pi = \{\pi_1, \ldots, \pi_{|\Pi|}\}$ as the set of stroke primitives (possible labels assigned to a stroke). Table 2 illustrates an example of a dataset of musical symbols, in which strokes are also labeled. In this case, $\Sigma = \{$𝅗𝅥, 𝅝$\}$ and $\Pi = \{st, wh, hn\}$.

A set of construction rules $R = \{(\sigma, (\pi_{i_1}, \ldots, \pi_{i_n})) : \sigma \in \Sigma, \pi_{i_j} \in \Pi\}_{i=1}^{|R|}$ can be obtained from the dataset of labeled musical symbols followed by their labeled sequence of strokes. That is, sequences of stroke primitives describing how specific musical symbols can be written. For instance, the dataset shown previously would be represented using the following set of rules:

$$\begin{array}{l} (\text{𝅝}, \text{wn}) \\ (\text{𝅗𝅥}, \text{hn}) \\ (\text{𝅗𝅥}, \text{st wn}) \end{array} \qquad (1)$$

It should be stressed that it is possible to find the same musical symbol

Table 2: Example of a dataset of pen-based musical symbols with labeled strokes.

Stroke Primitives (Π)		
stem (st)	white note-head (wn)	half note (hn)

Musical Symbols (Σ)	
Whole Note (𝅝)	Half Note (𝅗𝅥)

defined many times by the same sequence of stroke primitives. However, some stroke sequences are more likely to describe a musical symbol than others. For example, it might be more common for the *Half Note* (𝅗𝅥) to be written with two strokes (head plus stem) than with just one. It is, therefore, interesting to consider a prior probability for each rule in R. The prior probability of a rule $(\sigma, \bar{\pi}) \in R_\sigma$ will be denoted with $p(\bar{\pi}|\sigma)$. These probabilities will be estimated during the training stage.

Moreover, depending on the stroke labeling, it is possible that some music symbols might be defined by the same sequence of primitives. For example, if a user draws thin note-heads, the Half Note (𝅗𝅥) and the Quarter Note (♩) might be defined with the same strokes primitives: a stem plus a thin note-head. However, we will consider the use of a semantic language model so that the probability of each symbol will be different depending on its semantic meaning.

3.2. Input processing

The following lines describe how to build a finite-state machine that defines probabilities over the musical sequences that are permitted given a set of strokes received. This machine will be conditioned by both the input and the construction rules presented in the previous section, in addition to a formal language that restricts the machine to sequences that fulfill grammatical constraints.

3.2.1. Probability estimation

The input to the system is provided by a sequence of strokes $\bar{s} = (s_1, \ldots, s_{|\bar{s}|})$, in which each stroke is defined by an ordered sequence of 2D coordinates.

The first step involved in recognizing this input is knowing which types of strokes have actually been written. Since this process is not error-free, one way in which to approach it is by computing the probability of each received stroke being each stroke primitive considered.

Although several means can be employed to compute probabilities from labeled data, our estimation will be governed by dissimilarity functions between strokes. We denote this dissimilarity as $d(\cdot, \cdot)$. Our choice is justified by the fact that the stroke labeling is directed by a dissimilarity function. Furthermore, this paradigm is particularly suitable for interactive scenarios like that found in our task, as the simple addition of new prototypes to the training set is sufficient for incremental learning, whereas the size of the dataset can be controlled by dissimilarity-based data reduction algorithms (García et al., 2015).

In order to estimate a probability from a given dissimilarity, two different strategies are considered: Parzen Window and Nearest Neighbor.

- Parzen Window (Parzen, 1962) is a non-parametric technique used to estimate probability density functions from training samples. Given a series of samples $x_1, x_2, \ldots, x_n$ from an unknown distribution p, an estimated density $\hat{p}$ in a point x following Parzen Window method is

$$\hat{p}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h} \varphi \left(\frac{d(x, x_i)}{h} \right) \tag{2}$$

  The term φ refers to the *window function*, a symmetric function that integrates to one. The parameter h, called the bandwidth of the window, should be defined according to the volume of the region considered.

One of the main issues of the Parzen Window estimation is the choice of the window function φ. In practice, a standard Gaussian kernel is commonly assumed:

$$\varphi(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2} \tag{3}$$

- Nearest Neighbor: the problem of choosing the adequate window function φ can be avoided by considering only the nearest sample of the training data. This is called Nearest Neighbor estimation (Duda et al., 2001).

  Given a series of samples $x_1, x_2, \ldots, x_n$ from an unknown density function p, a common estimation for a point x becomes

$$\hat{p}(x) = \frac{1}{\min_{i=1}^{n} d(x, x_i)} \tag{4}$$

Note that a dissimilarity function between strokes is needed to make use of the previous probability density function estimators. The digital surface collects the strokes at a fixed sampling rate, signifying that each one may contain a variable number of 2D points. Several functions for the measurement of dissimilarity can be applied to this kind of data. Those considered in this work include:

- Dynamic Time Warping (DTW) (Sakoe & Chiba, 1990): a technique used to measure the dissimilarity between two time signals which may be of different duration. This technique was first used in speech recognition, although its use has been extended to other fields (Hartmann & Link, 2010; Kim et al., 2014).

- Edit distance with Freeman Chain Code (FCC): the sequence of points representing a stroke is converted into a string using a codification based on Freeman Chain Codes (Freeman, 1961). The common Edit distance (Levenshtein, 1966) between strings is then applied.

- Normalized Stroke (NS): the whole set of points of the stroke is normalized to a sequence of n points by means of an equally resampling technique. A stroke is therefore characterized by an n-dimensional feature vector of 2D coordinates. Given two vectors of this kind, an accumulated Euclidean distance between the points of the sequences can be computed.

- Edit distance for Consecutive Points (CP) (Rico-Juan & Iñesta, 2006): an extension of the edit distance for its use with sequences of consecutive points.

It should be noted that these functions take into account the direction of the strokes. That is, a stroke written inversely represents a different object despite having the same shape. Although this can be seen as a drawback, treating these strokes differently is, in most cases, known to be beneficial (Connell & Jain, 2001).

Each combination of probability estimation and dissimilarity functions will be evaluated empirically. Our intention is to both measure the robustness of our approach to these two elements and study the most appropriate choice for this task.

3.2.2. Building a Probabilistic Automaton

At this point, both the probability of each stroke being each stroke primitive and the musical symbol construction rules from stroke primitives are known. This knowledge can be merged to obtain a machine that defines the probability of sequences of musical symbols.

The machine to be built is a Probabilistic Finite-State Automaton (Pfa). A Pfa is a generative device for which there are a number of possible definitions (Paz, 1971; Vidal et al., 2005).

Definition 1. *A* Probabilistic Finite-State Transducer (Pfa) *is a tuple* $\mathcal{A} = \langle \Sigma, Q, \mathbb{I}, \mathbb{F}, \delta \rangle$*, where:*

- Σ *is the alphabet;*

- *Q =$\{q_1,\ldots,q_{|Q|}\}$ is a finite set of* states*;*

- $\mathbb{I}: Q \to \mathbb{R} \cap [0,1]$ *(initial probabilities);*

- $\mathbb{F}: Q \to \mathbb{R} \cap [0,1]$ *(final probabilities);*

- $\delta: Q \times \Sigma \times Q \to \mathbb{R} \cap [0,1]$ *is the complete transition function;* $\delta(q,a,q')=0$ *can be interpreted as "no transition from q to q' labeled with a".*

$\mathbb{I}$, δ and $\mathbb{F}$ are functions such that:

$$\sum_{q\in Q} \mathbb{I}(q) = 1, \tag{5}$$

and $\forall q \in Q$*,*

$$\mathbb{F}(q) + \sum_{a\in\Sigma,\ q'\in Q} \delta(q,a,q') = 1. \tag{6}$$

Given $x \in \Sigma^*$, an *accepting* x-path is a sequence $\gamma = q_{i_0}a_1q_{i_1}a_2\ldots a_nq_{i_n}$ in which $x = a_1\cdots a_n$, $a_i \in \Sigma$ and $\delta(q_{i_{j-1}}, a_j, q_{i_j}) \neq 0$, $\forall j$ such that $1 \leq j \leq n$. Let $\Gamma_{\mathcal{A}}(x)$ be the set of all paths accepting x. The probability of the path γ is defined as $Pr_{\mathcal{A}}(\gamma) = \mathbb{I}(q_{i_0}) \cdot \prod_{j=1}^{n} \delta(q_{i_{j-1}}, a_j, q_{i_j}) \cdot \mathbb{F}(q_{i_n})$ and the probability of the sequence x is obtained by obtaining the sum of the probabilities of all the paths in $\Gamma_{\mathcal{A}}(x)$.

Our Pfa is constructed as described in Algorithm 1. The machine generates as many states as there are strokes in the input plus 1. The ith state represents the fact that every stroke from the first until the $(i-1)$th has been processed (state 0 means that no stroke has yet been processed). This is why the only initial state is the first and the only final state is the last. For each state, the set of construction rules is queried and a new edge is created for every single rule. The length of each edge corresponds to the number of stroke primitives that contain the rule it represents. The label for the edge is given by the musical symbol of the rule. Note that those edges that would end beyond the last state will be discarded. Finally, the probability of these edges is given by both the product of the probability of the strokes being the primitives of the rule and

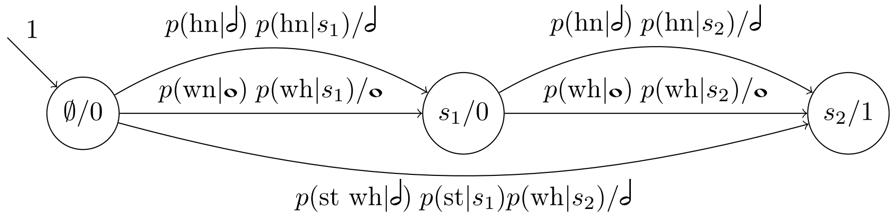


Figure 2: Example of PFA generated from an input of two strokes and the set of construction rules defined in Eq. 1. An arrow toward a state represents its initial probability (omitted when 0). Text inside states represents the last stroke processed and the probability of stopping. Text over edges represents the probability of the transition and its label.

the prior probability of the rule itself. The calculation of these probabilities was shown above, and it is therefore assumed that they are available when running Algorithm 1.

```
**Data**: $\bar{s} = (s_1, \ldots, s_{|\bar{s}|})$, $R = \{(\sigma, (\pi_{i_1}, \ldots, \pi_{i_n})) : \sigma \in \Sigma, \pi_{i_j} \in \Pi\}^{|R|}$
**Result**: $(Q, \Sigma, \mathbb{I}, \mathbb{F}, \delta)$ : PFA
$Q \leftarrow \{q_0, \ldots, q_{|\bar{s}|}\}$
$\mathbb{I}(q_0) \leftarrow 1$
$\mathbb{F}(q_{|\bar{s}|}) \leftarrow 1$
forall the $q_i \in Q$ do
    forall the $(\sigma, (\pi_{i_1} \ldots \pi_{i_n})) \in R$ do
        if $i + n \leq |\bar{s}|$ then
            $\delta(q_i, \sigma, q_{i+n}) \leftarrow p(\pi_1 \ldots \pi_n|\sigma) \prod_{k=0}^{n} p(\pi_{k+1}|s_{i+k})$
        end
    end
end
```
Algorithm 1: Building a PFA from an input sequence and the set of symbol construction rules.

Figure 2 shows an example of PFA given an input sequence $\bar{s} = s_1 s_2$ and the set of construction rules in Eq. 1. Although this is not the case, different paths may have the same probability depending on the set of rules.

3.2.3. Avoiding sequences that are not well-formed

The machine obtained in the previous section is able to define a probability for every sequence in Σ^*. However, it is clear that not all these sequences are *grammatically* correct. Hence, the next step is to ensure that only well-formed sequences have a non-null probability of being produced.

It is assumed that well-formed musical sequences can be defined by means of a regular language. It is thus possible to build a Deterministic Finite-State Automaton that only accepts those sequences that fulfill the constraints of the music.

Definition 2. *A* Deterministic Finite-State Automaton (DFA) *is a tuple* $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$ *where:*

- Q *is the set of states;*
- Σ *is the alphabet;*
- $\delta : Q \times \Sigma \rightarrow Q$ *is the transition function;*
- q_0 *is the initial state;*
- $F \subseteq Q$ *is the set of accepting states.*

A sequence $\bar{\sigma} \in \Sigma^*$ *is accepted by the* DFA *if, and only if,* $\delta^*(q_0, \hat{\sigma}) \in F$.

Let us once again consider a small alphabet of musical symbols $\Sigma = \{$𝅗𝅥$,$ 𝅝$\}$. Figure 3 shows a toy DFA which only accepts sequences that begin with a *Half note* (𝅗𝅥). Note that this language does not make any sense musically, but is used here as an easy example to guide the explanation.

The semantic information provided by the DFA can be merged with the previous PFA. Our goal is to change the probabilities so that sequences that do not belong to the language are nullified (zero probability). This machine is generated by computing the intersection of the DFA and the PFA. The output of this intersection is a new probabilistic machine for which sequences that do not belong to the language end in a non-final state, *ie.,* a state with a stop probability that is equal to 0.

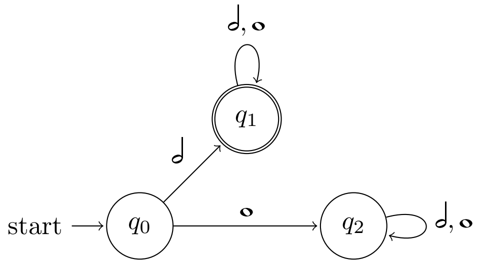


Figure 3: An example of Dfa that accepts only sequences whose first symbol is a *Half note*. Double circle indicates a final state.

The intersection between these machines can be obtained computationally by following Algorithm 2. Given a Pfa $\mathcal{A}$ and a Dfa $\mathcal{D}$, we obtain a new Pfa $\mathcal{B}$. This new machine has $Q_{\mathcal{A}} \times Q_{\mathcal{D}}$ states. We denote each of these states using a pair (q_a, q_d). The first element indicates the state of $\mathcal{A}$, while the second indicates the state of $\mathcal{D}$ that the new state represents. The initial probabilities of $\mathcal{B}$ are equal to those in $\mathcal{A}$ for every state that also represents an initial state in $\mathcal{D}$. Otherwise, probabilities are equal to 0. Similarly, the final probabilities of $\mathcal{B}$ are equal to those in $\mathcal{A}$ as long as the state is also in the set of final states of $\mathcal{D}$. Finally, the probability from a state (q_a, q_d) to $(q_{a'}, q_{d'})$ with a symbol σ is equal to the transition from q_a to $q_{a'}$ in $\mathcal{A}$ with this symbol as long as a transition from q_d to $q_{d'}$ with σ is allowed in $\mathcal{D}$. A post-processing step may normalize the new Pfa, thus enabling it to fulfill the conditions stated in Def. 1.

An example of this intersection is shown in Fig. 4. Unattainable states have been omitted for the sake of clarity. It should be stressed that the path (𝅝, 𝅗𝅥) now has the same probability as before, but the final probability of the sequence is 0 owing to the null stop probability of state (s_2, q_2). Useless paths could be removed to reduce the complexity of the machine generated.

3.3. Decoding strategies

At this point, our model is able to assign a probability to each valid sequence of musical symbols. The last step is to apply some kind of decoding strategy in order to output a hypothesis concerning the input sequence of strokes. The possibility of approaching this stage following different strategies is another

```
**Data**: $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \mathbb{I}_{\mathcal{A}}, \mathbb{F}_{\mathcal{A}}, \delta_{\mathcal{A}})$ : PFA, $\mathcal{D} = (Q_{\mathcal{D}}, \Sigma, \delta_{\mathcal{D}}, q_{\mathcal{D}_0}, F_{\mathcal{D}})$ : DFA
**Result**: $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, \mathbb{I}_{\mathcal{B}}, \mathbb{F}_{\mathcal{B}}, \delta_{\mathcal{B}})$ : PFA
$Q_{\mathcal{B}} \leftarrow Q_{\mathcal{A}} \times Q_{\mathcal{D}}$
forall the $q_b = (q_a, q_d) \in Q_{\mathcal{B}} : q_d = q_{\mathcal{D}_0}$ do
    $\mathbb{I}_{\mathcal{B}}(q_b) \leftarrow \mathbb{I}_{\mathcal{A}}(q_a)$
end
forall the $q_b = (q_a, q_d) \in Q_{\mathcal{B}} : q_d \in F_{\mathcal{D}}$ do
    $\mathbb{F}_{\mathcal{B}}(q_b) \leftarrow \mathbb{F}_{\mathcal{A}}(q_a)$
end
forall the $q_b = (q_a, q_d) \in Q_{\mathcal{B}}$ do
    forall the $q_{b'} = (q_{a'}, q_{d'}) \in Q_{\mathcal{B}}$ do
        forall the $\sigma \in \Sigma$ do
            if $\delta_{\mathcal{D}}(q_d, \sigma) = q_{d'}$ then
                $\delta_{\mathcal{B}}(q_b, \sigma, q_{b'}) \leftarrow \delta_{\mathcal{A}}(q_a, \sigma, q_{a'})$
            end
        end
    end
end
```

Algorithm 2: Building a new PFA from the information provided by a PFA and a DFA.

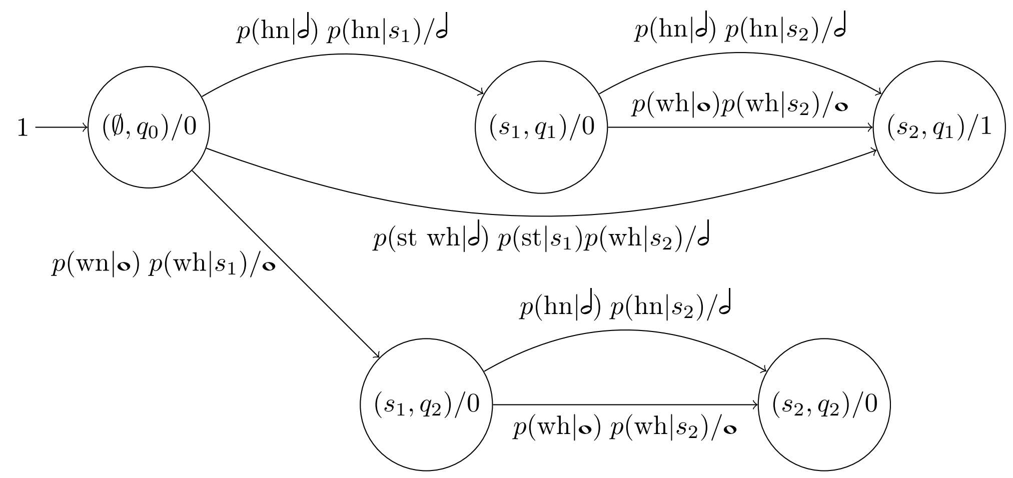

$p(\text{hn}|𝅗𝅥)\ p(\text{hn}|s_1)/𝅗𝅥$
$p(\text{hn}|𝅗𝅥)\ p(\text{hn}|s_2)/𝅗𝅥$
$p(\text{wh}|𝅝)p(\text{wh}|s_2)/𝅝$
1
$(\emptyset, q_0)/0$
$(s_1, q_1)/0$
$(s_2, q_1)/1$
$p(\text{st wh}|𝅗𝅥)\ p(\text{st}|s_1)p(\text{wh}|s_2)/𝅗𝅥$
$p(\text{wn}|𝅝)\ p(\text{wh}|s_1)/𝅝$
$p(\text{hn}|𝅗𝅥)\ p(\text{hn}|s_2)/𝅗𝅥$
$(s_1, q_2)/0$
$(s_2, q_2)/0$
$p(\text{wh}|𝅝)\ p(\text{wh}|s_2)/𝅝$

Figure 4: PFA obtained by the intersection of the PFA shown in Fig. 2 and the DFA shown in Fig. 3. An arrow toward a state represents its initial probability (omitted when 0). Text inside states represents the intersection of the original states from which it originates, along with the probability of stopping. Text over edges represents the probability of the transition and its label.

interesting advantage of our approach. The decoding strategies considered here are listed below:

- Most probable path (MPP): this seeks the most probable path of the PFA, and then outputs the sequence associated with this path. This is typically computed with an efficient Viterbi alike algorithm.

- Most probable sequence (MPS): searching for the most probable sequence of a distribution defined by a PFA is known to be NP-Hard (Casacuberta & de la Higuera, 2000). Nevertheless, a recent development makes its computation possible with a complexity of the inverse of the probability of the most probable sequence (de la Higuera & Oncina, 2014). This approach is used herein.

- Optimum decoding to minimize number of corrections (MNC): if it is assumed that the output will be corrected by a human supervisor, and that after each correction the machine will be allowed to output a new hypothesis (interactive approach), the optimum means of minimizing the expected number of sequential corrections is that of computing the algorithm developed by Oncina (2009).

All of these strategies will be compared experimentally.

3.4. Training stage

Having described the operation of the system, we shall now summarize the training stage that is performed. The system must learn from a set of labeled examples of isolated pen-based music symbols.

Let us again consider Σ as the set of musical symbols. Let $\mathcal{S}$ be the *stroke* space, using $\mathcal{S}^+$ as the (infinite) set of stroke sequences containing at least one stroke. Our training set T initially consist of pairs $\{(\bar{s}, \sigma : \hat{s} \in \mathcal{S}^+, \sigma \in \Sigma\}$, that is, sequences of one or more strokes associated with a musical symbol. Note that each symbol can be defined by a different number of strokes.

The first step is to obtain the set of stroke primitives. At this point, there is the open question of how to label each stroke in the dataset so as to group those that are more similar. It is clear that the alphabet of musical symbols is defined by the music notation itself, but this is not so in the case of the set of stroke primitives. The approach followed here is that shown in the work of Calvo-Zaragoza & Oncina (2015). This work proposes the use of a *k-medoids* clustering (Theodoridis & Koutroumbas, 2006) for the automatic labeling of strokes, given an *ambiguity* rate allowed. This rate defines the proportion of different musical symbols that can be defined by the same sequence of stroke primitives. The idea is to obtain the minimum number of stroke primitives without breaking the ambiguity rate. Since this value is a parameter to be tuned empirically, our experimentation will consider several possibilities for this rate.

As a result of this clustering process, we obtain a discrete set of stroke primitives, Π, and a function that maps each stroke onto its corresponding stroke primitive label. If each stroke is given a label, a set of construction rules can therefore easily be generated from T, as described in Sect. 3.1.

Having obtained the construction rules, it is now necessary to assign a prior probability to each one depending on how often it appears in the training set. Let $R_\sigma = \{(\sigma, \bar{\pi}) \in R : \sigma \in \Sigma, \bar{\pi} \in \Pi^+\}$ be the set of rules whose musical symbol is σ. The prior probability of a rule $(\sigma, \bar{\pi}) \in R_\sigma$ is then computed as:

$$p(\bar{\pi}|\sigma) = \frac{\#((\sigma, \bar{\pi}))}{\sum_{(\sigma,\bar{\pi}')\in R_\sigma} \#((\sigma, \bar{\pi}'))} \tag{7}$$

where $\#((\sigma, \bar{\pi}))$ denotes the number of times that a rule appears in the dataset. Note that $\sum_{(\sigma,\bar{\pi})\in R_\sigma} p(\bar{\pi}|\sigma) = 1$, for any $\sigma \in \Sigma$.

We also wish to map each input stroke onto the probability of it being each possible stroke primitive. As explained in Section 3.2, this probability estimation is governed by instance-based algorithms, signifying that there is no need for training and that it is merely necessary to query the set of strokes that are labeled after the clustering process.

4. Experimentation

This section describes the experimentation performed to assess the goodness of our proposal.

The HOMUS dataset (Calvo-Zaragoza & Oncina, 2014) of pen-based musical symbols will be used in this experimentation [1]. This set contains 15200 samples from 100 different musicians. We took advantage of the configuration of this set, and the series of experiments therefore consists of recognizing semi-synthetic pen-based music scores from two different scenarios: user-dependent, in which both learning and test data were provided by the same musician, and user-independent, in which all available data is mixed. The objective of the former scenario is to measure the performance when the user is known by the system, while that of the latter is to simulate a more general situation.

The input of both experiments is a series of 1000 sequences of musical symbols, generated randomly and which respect the language model defined. Each

[1] The dataset is freely available at `http://grfia.dlsi.ua.es/homus/`

sequence is used to generate a pen-based score using the data available from HOMUS. Figure 5 illustrates an example of this generation. The sequences generated contain 17.1 musical symbols, an average of ($\pm$ 3 of standard deviation), with a variable number of strokes depending on the synthetic generation.

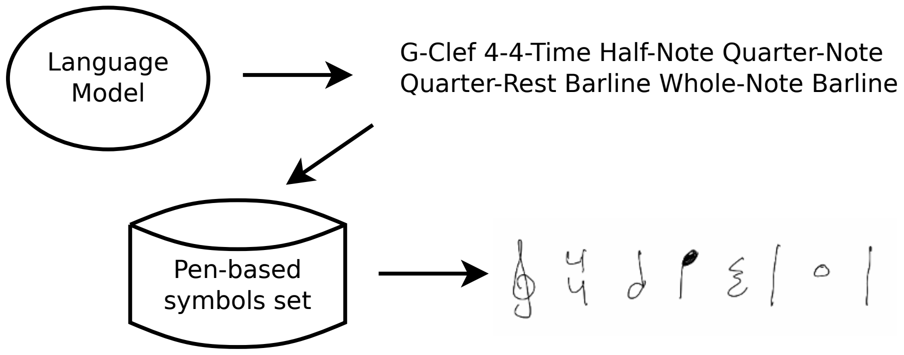

Language Model
G-Clef 4-4-Time Half-Note Quarter-Note Quarter-Rest Barline Whole-Note Barline
Pen-based symbols set

Figure 5: Example of the procedure used to generate semi-synthetic pen-based scores for experimentation.

The stroke labeling process is performed automatically, as mentioned in Section 3.1. We will allow different ambiguity rates, denoted as α, ranging from 0 to 0.3 depending on each scenario. Higher values of ambiguity are expected to give very poor results.

Please recall that, as mentioned in Section 3.2.3, it is necessary to develop a DFA that indicates which musical sequences are allowed. Since the development of a restrictive model for music notation is an extremely complex task, we shall restrict ourselves to taking into account the time-signature constraints of each bar in the composition. Note that this is an important source of disambiguation. For instance, similar symbols such as the Half Note (𝅗𝅥) or the Quarter Note (♩) would play a very different role for the time-signature constraints. AppendixA describes an efficient way in which to build such a model automatically. In these experiments, we specifically consider scores with a *common time* metric ($\frac{4}{4}$ or 𝄴), one of the most common in modern Western music.

With regard to the performance evaluation, there are different ways in which to measure the goodness of a hypothesis depending on the main goal pursued.

To this end, we take the following metrics of interest:

- Error rate (E_r): this provides the number of times (out of the total) that the first hypothesis provided by the system is the actual input. It is used when a 0/1 loss function is assumed.

- Average edit distance (E_d): this provides the expected number of corrections per sequence by computing the average edit distance between the solutions provided by the system and the actual inputs. It is the unnormalized version of the Word Error Rate.

- Average number of corrections (C): a good way in which to measure the performance of this kind of systems is to count the number of corrections the user would have to make before obtaining the sequence actually written (Vidal et al., 2008). After each correction, the system is allowed to recompute a new hypothesis, which is expected to be more accurate since some parts of the solution are known.

The results achieved in each scenario are presented in the following subsections. It is important to stress that we do not compare this experiment with those of other approaches. As stated in Section 2, no fully learning-based music notation recognition has yet been proposed, and a fair comparison is not therefore possible.

4.1. User-dependent experiment

The user-dependent experiment consists of the following steps: for every sequence, a user from the dataset is randomly selected and an automatic stroke primitive labeling is performed with different ambiguity rates ranging from 0 to 0.2. The data in each of them is split into a test and a train set. The test set is used to build the pen-based score that fulfills the definition of the sequence generated, whereas the train set is used to both extract the set of construction rules of musical symbols and train the stroke probability estimators (see Section 3.4). The machine obtained is then decoded to produce a hypothesis concerning the input sequence. Finally, the aforementioned performance evaluation is applied.

Table 3 shows the results attained after carrying out this experiment. Although further analysis is presented below, an initial remark to begin with is that the performance is quite promising. The best results yield that 90 % of these sequences are perfectly recognized ($E_r = 0.10$). From another point of view, the results show 0.3 mistakes per sequence (E_d) or 0.28 corrections needed per sequence (C). This implies that the post-processing user correction phase could be assumed effortlessly, regardless of the use of an interactive approach.

If the above is studied in greater detail, it will be noted that the allowable ambiguity rate in stroke labeling has a particular impact on the results. In most cases, the results of the same probability estimator with the same decoding strategy are noticeably worse as the ambiguity rate becomes higher. For instance, the NN estimation with FCC and decoded by MPS achieves an error rate of 0.10 with $\alpha = 0$, but error rates of 0.17 and 0.38 are obtained with $\alpha = 0.1$ and $\alpha = 0.2$, respectively. This tendency is depicted in Fig. 6, which shows the average results obtained using the different decoding strategies and the probability density estimator for each ambiguity rate considered. Note that the results degenerate as α becomes higher. A remarkable exception occurs in the case of the NN estimation with FCC when the number of corrections (C) is considered as an evaluation metric. In this case, the best result (which is also the best with regard to the whole experiment) is that with $\alpha = 0.1$.

Not surprisingly, stroke probability estimation is also a relevant factor. However, it would appear that the dissimilarity function is more accurate when compared to the probability density estimator. As a whole conclusion in this respect, NN with FCC appears to be the most accurate stroke probability estimator for this task. Furthermore, the results hardly vary as regards the decoding strategies considered, especially in the case of $\alpha = 0$. Despite this, the best average results for E_r and E_d are attained when using the MPS strategy, whereas MNC needs the lower number of corrections. The worst average results are, in most cases attained when using MPP.

Table 3: Mean results of the user-dependent experiment with an average number of 1000 random sequences of length 17.1. Several allowable ambiguity rates (α) for the automatic stroke labeling are considered.

Prob.	Dissim.	Decod.	$\alpha = 0$			$\alpha = 0.1$			$\alpha = 0.2$		
			E_r	E_d	C	E_r	E_d	C	E_r	E_d	C
NN	FCC	MPP	0.11	0.31	0.36	0.19	0.62	0.31	0.42	1.54	0.73
		MPS	**0.10**	**0.30**	0.36	**0.17**	**0.55**	0.28	**0.38**	**1.39**	0.65
		MNC	**0.10**	0.31	0.35	0.18	0.59	**0.26**	0.40	1.46	**0.62**
	DTW	MPP	0.16	0.56	0.45	0.32	1.12	0.55	0.54	2.08	1
		MPS	0.16	0.57	0.44	0.33	1.13	0.53	0.48	1.97	0.9
		MNC	0.17	0.58	0.44	0.32	1.13	0.52	0.51	2.04	0.91
	NS	MPP	0.19	0.70	0.60	0.36	1.38	0.63	0.54	2.37	1.09
		MPS	0.19	0.70	0.59	0.34	1.33	0.59	0.52	2.25	1.02
		MNC	0.19	0.69	0.58	0.34	1.32	0.58	0.54	2.37	1.05
	CP	MPP	**0.10**	**0.30**	0.51	0.23	0.8	0.37	0.45	1.87	0.81
		MPS	**0.10**	**0.30**	0.51	0.19	0.73	0.34	0.40	1.7	0.75
		MNC	**0.10**	**0.30**	0.51	0.2	0.75	0.35	0.43	1.8	0.77
Parzen	FCC	MPP	0.14	0.50	0.34	0.23	0.83	0.44	0.43	1.71	0.76
		MPS	0.13	0.50	0.33	0.2	0.74	0.38	0.39	1.57	0.68
		MNC	0.13	0.50	**0.31**	0.21	0.78	0.39	0.42	1.65	0.71
	DTW	MPP	0.27	1.01	0.79	0.49	2.11	1.14	0.65	3.06	1.60
		MPS	0.26	0.94	0.72	0.45	1.96	1.03	0.61	2.89	1.50
		MNC	0.26	0.94	0.70	0.45	1.92	0.99	0.65	2.94	1.46
	NS	MPP	0.19	0.72	0.52	0.37	1.56	0.79	0.57	2.69	1.29
		MPS	0.19	0.74	0.48	0.34	1.48	0.74	0.56	2.6	1.24
		MNC	0.19	0.74	0.48	0.34	1.51	0.75	0.57	2.66	1.22
	CP	MPP	0.17	0.57	0.48	0.29	1.05	0.54	0.46	1.9	0.88
		MPS	0.16	0.56	0.45	0.23	0.88	0.46	0.43	1.73	0.79
		MNC	0.16	0.59	0.46	0.25	0.96	0.45	0.45	1.8	0.8

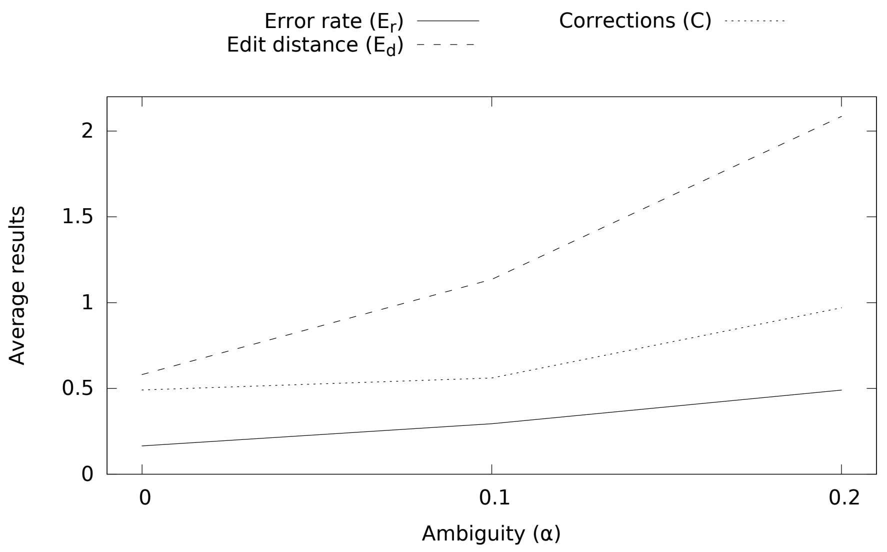


Figure 6: Impact of the stroke labeling ambiguity on the results attained. Average results for all decoding strategies and probability density estimators are shown with regard to the ambiguity rate allowed.

4.2. User-independent experiment

The user-independent experiment is a more general scenario in which the system does not know what kind of handwriting style it will receive and must, therefore, learn from samples of many different musicians. Given the large number of different strokes found in this scenario, the automatic labeling of strokes during the training set fails to obtain a non-ambiguous clustering when using less than 500 labels (maximum considered). The experiments are, therefore, shown with ambiguity rates of 0.1, 0.2 and 0.3.

Table 4 shows the results obtained in this experiment. As a general analysis, the results are worse than in the previous case, since the complexity of the task is higher. However, the best values obtained are still quite competitive: 64 % of the sequences were perfectly recognized with the first hypothesis; on average, only 1.10 of editing operations are necessary, with only 0.54 corrections when considering an interactive case. These figures demonstrate that the use of an interactive approach is highly profitable in this scenario, unlike that which occurred in the previous one.

Once again, the ambiguity rate would appear to be the most important factor as regards recognition (Fig.7 shows the average tendency with regard to this factor). Since a null ambiguity rate has not been possible, the best case in the user-independent scenario is still unknown. However, the results obtained using $\alpha = 0.1$ are at the same level as those obtained using $\alpha = 0.2$ in the user-dependent case. This leads us to believe that the user-independent scenario would not be very far from the user-dependent scenario if an optimal stroke labeling were performed.

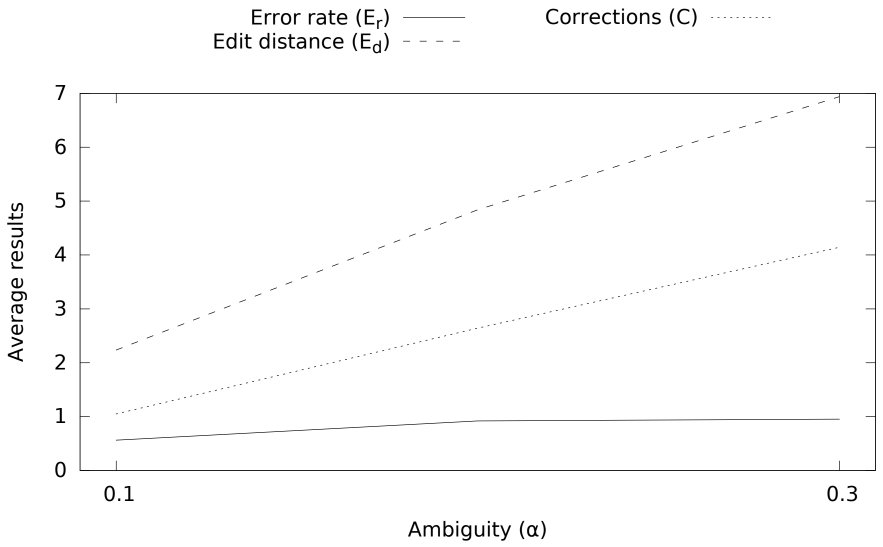


Figure 7: Impact of the stroke labeling ambiguity on the results attained. Average results among all decoding strategies and probability density estimators are shown with regard to the ambiguity rate allowed.

Furthermore, probability estimation has a higher impact in this scenario. For instance, the Parzen estimation using DTW attains fairly poor results in all the cases considered. Depending on each particular case, NN using FCC or Parzen using NS would appear to be the best configurations.

The decoding algorithm has less relevance in the results, as occurred in the previous scenario. Nevertheless, MPS generally achieves slightly better results

Table 4: Mean results of the user-independent experiment with 1000 random sequences of an average length of 17.1. Several allowable ambiguity rates (α) for the automatic stroke labeling are considered.

Prob.	Dissim.	Decod.	$\alpha = 0.1$			$\alpha = 0.2$			$\alpha = 0.3$		
			E_r	E_d	C	E_r	E_d	C	E_r	E_d	C
NN	FCC	MPP	0.39	1.52	0.59	0.94	4.70	2.37	0.94	6.81	4.11
		MPS	**0.36**	**1.10**	0.56	0.85	4.10	2.23	0.92	6.50	3.55
		MNC	**0.36**	1.13	**0.54**	0.89	4.32	2.20	0.93	6.60	3.55
	DTW	MPP	0.59	2.04	1.07	0.94	4.78	2.76	0.98	7.43	4.64
		MPS	0.54	1.90	0.90	0.90	4.58	2.46	0.95	7.04	4.12
		MNC	0.56	2.00	0.90	0.91	4.58	2.46	0.97	7.11	4.07
	NS	MPP	0.57	2.18	0.91	0.90	4.73	2.58	0.96	7.06	4.32
		MPS	0.55	2.13	0.88	0.94	4.91	2.50	0.94	6.60	4.07
		MNC	0.56	2.13	0.88	0.94	4.91	2.51	0.95	6.68	4.04
	CP	MPP	0.54	2.18	0.93	0.92	4.29	2.40	0.95	6.57	3.86
		MPS	0.54	2.26	0.91	0.90	4.11	2.18	0.93	6.59	3.73
		MNC	0.54	2.28	0.90	0.91	4.16	2.17	0.93	6.59	3.73
Parzen	FCC	MPP	0.68	2.90	1.31	0.92	5.13	2.83	0.97	6.94	4.45
		MPS	0.48	1.75	0.85	0.91	4.31	2.32	0.95	**6.37**	**3.62**
		MNC	0.50	1.76	0.84	0.91	4.36	2.31	0.95	**6.37**	**3.62**
	DTW	MPP	0.90	4.75	2.62	1.00	6.73	4.13	0.98	7.86	5.68
		MPS	0.79	3.53	1.71	0.97	6.05	3.39	0.98	7.73	4.78
		MNC	0.79	3.54	1.69	0.97	6.05	3.38	0.98	7.73	4.74
	NS	MPP	0.45	1.57	0.73	0.91	4.93	2.74	0.94	6.95	4.20
		MPS	0.37	1.16	0.62	**0.83**	**3.90**	2.33	0.94	6.60	3.64
		MNC	0.37	1.19	0.60	0.85	4.39	**2.32**	0.94	6.60	3.64
	CP	MPP	0.76	3.58	1.71	0.96	5.89	3.33	0.98	7.67	5.02
		MPS	0.66	2.53	1.30	0.96	5.10	2.76	0.96	7.07	4.11
		MNC	0.67	2.53	1.23	0.96	5.16	2.75	0.96	7.07	4.11

for E_r and E_d, whereas MNC does so for C.

5. Conclusions

This work presents a new approach for the recognition of pen-based music compositions using stroke similarity algorithms and finite-state machines. Our approach is able to learn writing styles from data, and notation is not therefore restricted to predefined rules or gestures.

The series of strokes received as input is mapped onto stroke primitive probabilities with similarity-based probability density estimators. These probabilities are combined with a set of learned rules describing musical symbols in terms of these primitives. A probabilistic model is then obtained, which is eventually crossed with a formal language in order to avoid those sequences that do not make musical sense. Finally, a decoding algorithm is applied to produce a hypothesis as a solution to the task.

Comprehensive experimentation has been carried out during which several metrics of interest have been evaluated considering a number of probability density estimators, stroke similarity functions and decoding strategies. Two main scenarios were considered: user-dependent and user-independent.

As expected, better recognition results were obtained from the user-dependent experiment, whose best results yielded that only 10 % of the hypotheses were wrong, whereas the other only needed a few corrections (an average of under 0.3). However, the user-independent scenario also provided competitive results, obtaining only 36 % of erroneous hypotheses, and otherwise requiring an average of around 1 of corrections (0.5 in the interactive case).

The accuracy of the recognition was found to be closely related to the degree of allowed ambiguity in the automatic stroke labeling process. One option to be considered is that of improving this process. In fact, the method used was unable to obtain a non-ambiguous stroke labeling in the user-independent scenario, and there is, therefore, still room for improvement. The dissimilarity measure utilized also proved, to a lesser extent, to be an important parameter to consider.

In this respect, NN estimation using FCC dissimilarity would appear to be the best choice in a broad sense.

As an overall conclusion, we have demonstrated that it is possible to develop a pen-based music recognition system in which the handwriting style is learned, in contrast to previous strategies based on hand-engineered heuristics. Moreover, the use of statistical models based on finite state machines has been justified, since they allow a proper decoding strategy for the optimization of each performance measure considered. The system has also proven to be robust: although the probability estimation has some influence on the results, all the possibilities performed competitively.

As future work, our interest lies in developing an interactive system in which user corrections will be used to continuously improve the performance of the system. Note that the training stage is completely data-driven and uses instance-based recognition algorithms. The mere inclusion of new examples in the training set is, therefore, sufficient as regards performing incremental learning. The main concern, however, is to provide a transparent and user-friendly means to receive feedback while the user is using the system, and how this feedback can be efficiently exploited. This should be one of the main objectives when making future improvements. A usable front-end application that takes advantage of the research explained here is also being considered.

Acknowledgements

This work was supported by the Spanish Ministerio de Educación, Cultura y Deporte through a FPU Fellowship (Ref. AP2012–0939) and the Spanish Ministerio de Economía y Competitividad through the TIMuL Project (No. TIN2013-48152-C2-1-R, supported by UE FEDER funds).

References

Anstice, J., Bell, T., Cockburn, A., & Setchell, M. (1996). The design of a pen-based musical input system. In *Proceedings of the 6th Australian Conference* *on Computer-Human Interaction* (pp. 260–267).

Bainbridge, D., & Bell, T. (2001). The challenge of optical music recognition. *Computers and the Humanities*, *35*, 95–121.

Calvo-Zaragoza, J., & Oncina, J. (2014). Recognition of pen-based music notation: The HOMUS dataset. In *Proceedings of the 22nd International Conference on Pattern Recognition (ICPR)* (pp. 3038–3043).

Calvo-Zaragoza, J., & Oncina, J. (2015). Clustering of strokes from pen-based music notation: An experimental study. In *Proceedings of the 7th Iberian Conference on Pattern Recognition and Image Analysis* (pp. 633–640).

Casacuberta, F., & de la Higuera, C. (2000). Computational Complexity of Problems on Probabilistic Grammars and Transducers. In *Proceedings of the 5th International Colloquium on Grammatical Inference: Algorithms and Applications* ICGI 2000 (pp. 15–24). London, UK, UK: Springer-Verlag.

Connell, S. D., & Jain, A. K. (2001). Template-based online character recognition. *Pattern Recognition*, *34*, 1–14.

Dalitz, C., Droettboom, M., Pranzas, B., & Fujinaga, I. (2008). A comparative study of staff removal algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *30*, 753–766.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern Classification*. (2nd ed.). New York, NY: John Wiley & Sons.

Fornés, A., & Sánchez, G. (2014). Analysis and recognition of music scores. In *Handbook of Document Image Processing and Recognition* (pp. 749–774). Springer.

Freeman, H. (1961). On the encoding of arbitrary geometric configurations. *IRE Transactions on Electronic Computers*, *EC-10*, 260–268.

García, S., Luengo, J., & Herrera, F. (2015). *Data Preprocessing in Data Mining* volume 72 of *Intelligent Systems Reference Library*. Springer.

George, S. E. (2003). Online pen-based recognition of music notation with artificial neural networks. *Computer Music Journal*, *27*, 70–79.

Hartmann, B., & Link, N. (2010). Gesture recognition with inertial sensors and optimized DTW prototypes. In *IEEE International Conference on Systems Man and Cybernetics (SMC)* (pp. 2102–2109).

de la Higuera, C., & Oncina, J. (2014). Computing the Most Probable String with a Probabilistic Finite State Machine. In *11th International Conference on Finite-State Methods and Natural Language Processing* (pp. 1–8).

Hopcroft, J. E. (1979). *Introduction to automata theory, languages, and computation*. Pearson Education India.

Kim, D.-W., Lee, J., Lim, H., Seo, J., & Kang, B.-Y. (2014). Efficient dynamic time warping for 3D handwriting recognition using gyroscope equipped smartphones. *Expert Systems with Applications*, *41*, 5180–5189.

Kim, J., & Sin, B.-K. (2014). Online handwriting recognition. In D. Doermann, & K. Tombre (Eds.), *Handbook of Document Image Processing and Recognition* (pp. 887–915). Springer London.

Lee, K. C., Phon-Amnuaisuk, S., & Ting, C. Y. (2010a). A comparison of hmm, naïve bayesian, and markov model in exploiting knowledge content in digital ink: A case study on handwritten music notation recognition. In *Multimedia and Expo (ICME), 2010 IEEE International Conference on* (pp. 292–297). IEEE.

Lee, K. C., Phon-Amnuaisuk, S., & Ting, C.-Y. (2010b). Handwritten music notation recognition using HMM – a non-gestural approach. In *Proceedings of the International Conference on Information Retrieval Knowledge Management (CAMP)* (pp. 255–259).

Levenshtein, V. (1966). Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, *10*, 707.

Liu, C., Yin, F., Wang, D., & Wang, Q. (2013). Online and offline handwritten Chinese character recognition: Benchmarking on new databases. *Pattern Recognition*, *46*, 155–162.

Macé, S., Éric Anquetil, & Couasnon, B. (2005). A generic method to design pen-based systems for structured document composition: Development of a musical score editor. In *Proceedings of the First Workshop on Improving and Assesing Pen-Based Input Techniques* (pp. 15–22). Edinghburg.

Miyao, H., & Maruyama, M. (2004). An online handwritten music score recognition system. In *Proceedings of the 17th International Conference on Pattern Recognition (ICPR)* (pp. 461–464). volume 1.

Miyao, H., & Maruyama, M. (2007). An online handwritten music symbol recognition system. *International Journal of Document Analysis and Recognition*, *9*, 49–58.

Mohri, M., Pereira, F., & Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, *16*, 69–88.

Mohri, M., Pereira, F., & Riley, M. (2005). Weighted automata in text and speech processing. *CoRR*, *abs/cs/0503077*.

Mondal, T., Bhattacharya, U., Parui, S. K., Das, K., & Roy, V. (2009). Database generation and recognition of online handwritten bangla characters. In *Proceedings of the International Workshop on Multilingual OCR* MOCR '09 (pp. 9:1–9:6). New York, NY, USA: ACM.

Ng, E., Bell, T., & Cockburn, A. (1998). Improvements to a pen-based musical input system. In *Proceedings of the Computer Human Interaction Conference* (pp. 178–185).

Oncina, J. (2009). Optimum algorithm to minimize human interactions in sequential Computer Assisted Pattern Recognition. *Pattern Recognition Letters*, *30*, 558–563.

Parzen, E. (1962). On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, *33*, 1065–1076.

Paz, A. (1971). *Introduction to probabilistic automata*. New York: Academic Press.

Plamondon, R., & Srihari, S. N. (2000). On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*, 63–84.

Poláček, O., Sporka, A. J., & Slavík, P. (2009). Music alphabet for low-resolution touch displays. In *Proceedings of the International Conference on Advances in Computer Enterntainment Technology* ACE '09 (pp. 298–301). New York, NY, USA: ACM.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*, 257–286.

Rebelo, A., Fujinaga, I., Paszkiewicz, F., Marcal, A., Guedes, C., & Cardoso, J. (2012). Optical music recognition: state-of-the-art and open issues. *International Journal of Multimedia Information Retrieval*, *1*, 173–190.

Rico-Juan, J. R., & Iñesta, J. M. (2006). Edit distance for ordered vector sets: A case of study. In D.-Y. Yeung, J. Kwok, A. Fred, F. Roli, & D. de Ridder (Eds.), *Structural, Syntactic, and Statistical Pattern Recognition* (pp. 200–207). Springer Berlin Heidelberg volume 4109 of *Lecture Notes in Computer Science*.

Sakoe, H., & Chiba, S. (1990). Dynamic programming algorithm optimization for spoken word recognition. In A. Waibel, & K.-F. Lee (Eds.), *Readings in speech recognition* (pp. 159–165). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Theodoridis, S., & Koutroumbas, K. (2006). *Pattern Recognition, Third Edition*. Orlando, FL, USA: Academic Press, Inc.

Vidal, E., Rodríguez, L., Casacuberta, F., & García-Varea, I. (2008). Interactive pattern recognition. In A. Popescu-Belis, S. Renals, & H. Bourlard (Eds.), *Machine Learning for Multimodal Interaction* (pp. 60–71). Springer Berlin Heidelberg volume 4892 of *Lecture Notes in Computer Science*. doi:`10.1007/978-3-540-78155-4_6`.

Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., & Carrasco, R. C. (2005). Probabilistic finite-state machines-part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, *27*, 1013–1025.

AppendixA. Automatic generation of regular languages for music sequences

Formal languages are widely used to improve sequential recognition systems, for which the output is a sequence of symbols. These models focus on detecting which hypotheses are less likely based on their context, or even which of them are not feasible owing to task constraints. The approach presented in this work makes use of a Deterministic Finite-State Automaton representation (DFA), which is able to define any Regular Language (RL).

The key aspect in a musical sequence, if it is to be grammatically correct, is related to the time signature. The time signature of a music score determines how many beats each note represents and how many beats each bar can contain. An approach with which to generate RLs with regard t to a given time signature in the form of a DFA is presented below. The algorithm is driven by a dynamic programming (DP) scheme that makes the process efficient.

The main problem when defining languages for musical sequences concerns the semantics of the symbol *dot* ($\cdot$). This symbol is placed to the right of some figures and increases their lengths by a factor of 1.5. That is, if the symbol represents two beats, it will represent three with the *dot*. During the construction of the DFA, this forces us to create paths with a memory that takes into account the last symbol processed in order to discover how many beats a subsequent *dot* gets.

We use Σ to define the set of musical symbols that may appear in our task. However, it should be noted that the *dot* symbol cannot appear behind any other symbol, but only behind those that represent tone and beat. We use $\Sigma_{\text{dot}} \subset \Sigma$ to define this subset. It is assumed that the *barline* ($|$) is a special symbol that only appears when all beats are got. We also define the function $T(\sigma)$, which provides the number of beats that gets any symbol $\sigma \in \Sigma$.

The proposed scheme is presented in algorithm 3. The maximum number of beats in a bar is calculated as the number of allowed beats (parameter *beats*) divided by the minimum number of beats got by any symbol (which defines the resolution allowed). The matrix of the DP stores states of the DFA and it is 2-dimensional: one dimension is used to indicate the beats got so far, while the other is used to indicate the last symbol received. This is used to discover how many beats the subsequent *dot*would get. Symbol ϵ denotes a symbol that does not belong to Σ_{dot}, and *dot* is not therefore possible directly after.

The DFA generated starts in the state stored in $PD[0, \epsilon]$. For the sake of simplicity, it is assumed that the information needed in the recursive procedure is accessed globally. During the recursive action (function f), the following steps are computed:

1. If the state has already been visited, the function returns its value stored in the DP matrix. Otherwise, a new state is created.
2. If the last symbol belongs to Σ_{dot}, a *dot* transition from the new state is generated, as long as this is possible, taking into account the number of remaining beats allowed.
3. For any possible symbol, an out-transition is, if possible, generated. If the symbol belongs to Σ_{dot}, the recursive call includes the symbol. Otherwise, ϵ is included.
4. Before returning the new state created, it is stored in the DP matrix in order to save future repetitions of the same call.

After the recursive procedure has ended, all the possible sequences that getting the maximum number of beats are connected to the starting state by means of a *barline* transition.

As an example, Fig. A.8 illustrates a DFA generated for a time signature of $\frac{4}{4}$, considering $\Sigma = \{$𝅗𝅥, 𝄼, ♩, 𝄽$\}$ and $\Sigma_{\text{dot}} = \{$𝅗𝅥$\}$.

Note that this DFA is focused solely on the bars. An additional process must pay attention to many other features of the musical scores. For instance:

- Include some states at the beginning such that sequences must contain a clef, the possibility of having a key signature and the time signature symbol. The last of these states directly connects to the first state generated by Algorithm 3.

```
Algorithm getDFA(beats, T, Σ, Σ_dot)
    $M = \frac{\text{beats}}{\min_{\sigma \in \Sigma_{\text{dot}}} T(\sigma)}$
    $PD : M \times \Sigma_{\text{dot}} \cup \{\epsilon\} \to \text{State}$
    $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$
    $Q = \emptyset$
    $q_0 = \texttt{f}(0,\epsilon)$
    $F = \{q_0\}$
    forall the $\sigma \in \Sigma_{dot} \cup \{\epsilon\}$ do
        $\delta(PD[M,\sigma], |) = q_0$
    end
    return $\mathcal{D}$
```

```
Procedure f(time, last)
    if $PD[time,last] \neq \emptyset$ then
        return $PD[\text{time,last}]$
    end
    State q
    $Q \to Q \cup \{q\}$
    if $last \in \Sigma_{dot}$ then
        if $time + \frac{T(last)}{2} \leq M$ then
            $\delta(q, \cdot) = f(time + \frac{T(last)}{2}, \epsilon)$
        end
    end
    forall the $\sigma \in \Sigma$ do
        if $time + T(\sigma) \leq M$ then
            if $\sigma \in \Sigma_{dot}$ then
                $\delta(q,\sigma) =$
                $f(time + T(\sigma), \sigma)$
            else
                $\delta(q,\sigma) =$
                $f(time + T(\sigma), \epsilon)$
            end
        end
    end
    $PD[\text{time,last}] = \text{q}$
    return q
```

Algorithm 3: Dynamic programming scheme used to generate a DFA fulfilling a given time signature (defined by *beats* and T).

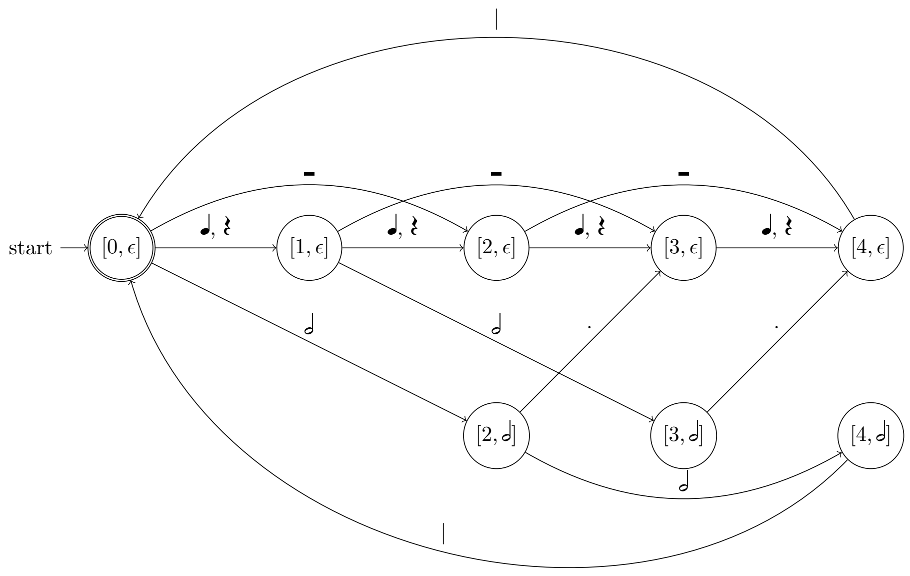


Figure A.8: Example of an automatically generated Dfa. Text inside states indicate from which position in the PD matrix they originate.

- Ensure that accidentals are placed only before a symbol with pitch. This also prevents the infinite repetition of those symbols that do not get beats.