Exploiting the Tradeoff Between Precision and Cpu-time to Speed Up Nearest Neighbor Search

Pierre Roy, Jean-Julien Aucouturier, François Pachet and Anthony Beurivé

SONY Computer Science Laboratory Paris
6, rue Amyot 75005 Paris, France.
{roy,jj,pachet,beurive}@csl.sony.fr

ABSTRACT

We describe an incremental filtering algorithm to quickly compute the N nearest neighbors according to a similarity measure in a metric space. The algorithm exploits an intrinsic property of a large class of similarity measures for which some parameter p has a positive influence both on the precision and the cpu cost (*precision-cputime tradeoff*). The algorithm uses successive approximations of the measure to compute first cheap distances on the whole set of possible items, then more and more expensive measures on smaller and smaller sets. We illustrate the algorithm on the case of a timbre similarity algorithm, which compares gaussian mixture models using a Monte Carlo approximation of the Kullback-Leibler distance, where p is the number of points drawn from the distributions. We describe several Monte Carlo algorithmic variants, which improve the convergence speed of the approximation. On this problem, the algorithm performs more than 30 times faster than the naive approach.

Keywords: Nearest Neighbor, Similarity Measure, Timbre, Large Databases.

1 INTRODUCTION

Algorithms for fast nearest neighbor (NN) searching in general metric spaces are of considerable interest for content-based retrieval in large music databases. Answering NN queries requires computing the relative distance between complex data objects, such as songs in audio or symbolic format, which is typically a very costly operation.

1.1 Metric Space Index Structures

One approach for speeding-up NN search is to use pre-built index structures. Traditional index structures, such as B+-trees or KD-trees (Samet, 1989), only work for datasets which can be represented in a suitable vector (euclidean) space, i.e. for which there exists an ordering of the data that preserves relative similarities. This holds for a number of simple similarity functions, e.g. those based on the euclidean distance between feature vectors computed from audio (Wold and Blum, 1996). In Reiss et al. (2001), we reviewed a number of such indexing techniques in the context of Music Information Retrieval. However, a large number of similarity algorithms are not suitable to such vector-space index structures, as they only provide a distance function or metric to measure the dissimilarity between data points. This is notably true of similarity functions based on statistical pattern recognition, such as timbre similarity (Aucouturier and Pachet, 2004) where songs are modeled with statistical models and the models compared with distribution comparison measures such as Kullback-Leibler or transportation distances.

There has been a number of proposals for metric-space index structures, most of which exploit the triangle-inequality property of the distance function to prune distance calculations during searching. Obviously, one may first have to verify that the distance measure indeed verifies the triangle inequality, as e.g. Vidal et al. (1988) for the edit-distance. The *Approximating and Eliminating Search Algorithm* (AESA) (Juan et al., 1998) performs NN search in approximately constant average-time, at the expense of pre-processing the matrix of pairwise distances between objects. This is typically well suited for matching incoming objects against a small (a few thousand) dataset of pre-computed prototypes, e.g. isolated word recognition tasks. However, as the space requirements of typical music databases keep increasing (for instance, the SONY Connect service[1] offers more than 700,000 tracks as of 2004), computing the whole N^2 similarity matrix is simply not feasible. The M-tree (Ciacca et al., 1997) and the Multi-vantage point (MVP) tree (Bozkaya and Ozsoyoglu, 1999) are radius-based indexing methods that do not require the computation of the whole similarity matrix, while still preserving fast access time. In these structures, the data is hierarchically organized in clusters defined by a center and a radius (the maximum distance from the center to any point in the cluster). If a query is too far from the center of a cluster, by virtue of the triangle inequality, all the points within the cluster can be pruned, and the corre-

[1]http://www.connect.com

sponding distances calculations can be spared. Miranker et al. (2003) have used, compared and improved such techniques in the context of large image and biological protein structures databases. A recent application of vantage point indexing to melodic similarity in music databases can be found in Typke et al. (2003).

1.2 Exploiting the Tradeoff Between Precision and Cputime

In this paper, we propose a generic algorithm for fast NN search in metric spaces which relies neither on an index structure[2], nor on the verification of the triangle inequality by the distance measure. The algorithm exploits an intrinsic property of a large class of similarity algorithms, which exhibit a *precision-cputime tradeoff* for some parameter p (*tradeoff parameter*), i.e. for which both the precision and the cputime increase with p.

Many music similarity measures proposed in the literature exhibit this precision-cputime tradeoff. This is notably true for pattern recognition distance measures, such as Foote (1997); Welsh et al. (1999); Pampalk et al. (2003); Aucouturier and Pachet (2004). In such measures, the distributions of each song's frame-based feature vectors (e.g. Mel-Frequency Cepstrum Coefficients MFCCs) are modeled (e.g. with Gaussian Mixture Models GMMs) then compared (e.g. using Kullback-Leibler). Many candidates exist for the tradeoff parameter p:

- p may be the size of the feature vector. For instance, the number of MFCCs typically influences the precision of the measure (as illustrated e.g. in Aucouturier and Pachet (2004)), but also the dimension of the model, hence the cpu time both for learning and comparing.
- p may also be the size of the model, e.g. the number of gaussian components in a GMM, or the bin size of a histogram. The more complex the model, the more precise the measure[3], but also the more expensive the learning and the comparison.
- p can also be found at the model comparison stage. In Section 3, we consider a Monte-Carlo sampling approximation of the Kullback Leibler distance between GMMs: the more samples are drawn from the GMMs, the more precise is the approximation by virtue of the central limit theorem, but also the more expensive are both the sampling and the distance computation.

We propose to exploit the precision-cputime tradeoff of such distance algorithms $\mathcal{A}$ to efficiently calculate the result of NN queries. We use n successive refinements of $\mathcal{A}$ to compute first cheap, unprecise distances (i.e. $\mathcal{A}(p)$ for p small) on the whole set of possible items, then more and more expensive and precise distances (i.e. $\mathcal{A}(p)$ for p big) on smaller and smaller sets. If the precision $PREC(p)$ of the distance measure increases *faster*[4] than the cputime $CPU(p)$, then we will show that the cumulated cpu time of the successive steps using $\mathcal{A}(p_0)$, $\mathcal{A}(p_1)$, ..., $\mathcal{A}(p_{n-1})$ may be a lot smaller than the direct computation of the most precise distance $\mathcal{A}(p_{n-1})$ on the whole set of items.

This approach can be viewed and implemented as a planning wrap-up around an existing distance measure, to speed up the associated nearest neighbor search. We show that dramatic speed-up can be achieved without modifying the implementation of the underlying distance measure.

Section 2 presents a general formulation of the algorithm, discusses the required condition on the distance measure, and explains the optimization process to find the optimal sequence of steps that yields the smallest total cputime. In Section 3, we illustrate the algorithm on the case of NN queries, using a timbre similarity algorithm (Aucouturier and Pachet, 2004), which compares Gaussian mixture models using a Monte Carlo approximation of the Kullback-Leibler distance, where the tradeoff parameter p is the number of points drawn from the distributions. We describe several Monte Carlo algorithmic variants, which improve the convergence speed of the approximation, and report speed improvement factors as high as 30.

2 INCREMENTAL FILTERING

In Section 1.2, an algorithm for fast NN searching in metric spaces was sketched out. This algorithm can be seen as a particular illustration of a more general approach which we present in this section.

In this more general context, we are interested in computing the elements of a set $\mathcal{S}$ that satisfy a given criterion c, called the target criterion. Note that computing the NN of a given item with respect to a given measure is a particular instance of this schema. The approach we present applies to any target criterion that can be approximated by a series of criteria with the following property: rough approximations of the target criterion are easy (fast) to compute, whereas good (precise) approximations of the target criterion take longer. Moreover, approximations are faster to compute than the target criterion itself.

The standard approach to computing the set N of elements of $\mathcal{S}$ that satisfy c is to evaluate c against every item in $\mathcal{S}$, retaining only those items that satisfy c. Roughly speaking, our approach consists in starting with a first criterion that can be evaluated quickly, to eliminate irrelevant items, and then, to progressively evaluate criteria that are better approximations of the target criterion, finishing with the target criterion itself, to achieve the task. The idea behind this strategy is that if the precision of the successive criteria increases faster than their computation cost, we can save a substantial amount of computation time, because criteria that are expensive to evaluate will be evaluated against fewer items.

2.1 Definitions and Assumptions

Let us first introduce some necessary definitions and conventions:

- $\mathcal{S}$ is a finite set.

[2]therefore, it is compatible and complementary with the above-described metric index structures

[3]this is not taking into account the curse of dimensionality, see Bishop (1995)

[4]in a sense to be defined in Section 2

$$\begin{tikzcd}
S=N_0 \arrow[rrrr, "c"] & & & & N \\
S=N_0 \arrow[r, "c_0"] & N_1 \arrow[r, "c_1"] & N_2 \arrow[r, dashed] & N_n \arrow[r, "c_n = c"] & N
\end{tikzcd}$$

Figure 1: Instead of computing N directly by applying c, we iteratively compute the N_i for $i = 0, 1, ..., n$

- c is a criterion defined over $\mathcal{S}$

$$c : S \rightarrow \{true, false\} \tag{1}$$

  We call c the *target* criterion.

- c_0, c_1, ..., c_n are criteria defined over $\mathcal{S}$ that approximate c with increasing precision, with the convention that $c_n = c$.
- N is the subset of $\mathcal{S}$ containing those elements that satisfy c. The goal of the algorithm is to compute N.
- Similarly, N_i is the subset of $\mathcal{S}$ that contains those elements that satisfy c_{i-1}. By convention, we define $N_0 = \mathcal{S}$.
- $t(c_i) < t(c_{i+1}) \forall i \in [0, n-1]$ where $t(c_i)$ denotes the cpu time needed to compute $c_i(x)$ for any element $x \in \mathcal{S}$.

Note that the two following properties are a formalization of the type of algorithms described in Section 1.2 (precision-cputime tradeoff)

Property 1 c_0, c_1, ..., c_n *approximate* c *with increasing precision*

Property 2 *The cost of computing* c_i *increases with* i*, i.e.* $t(c_{i+1}) > t(c_i)$

The algorithm can be described by a simple idea, illustrated in Figure 1: instead of computing N directly by applying c, we iteratively compute the N_i for $i = 0, 1, ..., n$.

Property 3 $N_n \subseteq N_{n-1} \subseteq ... \subseteq N_1 \subseteq N_0 = \mathcal{S}$ *(i.e.* $c_{i+1} \Rightarrow c_i$*)*

When Property 3 holds on the N_i sets, it is straightforward to show that $c_i(N_i) = c_i(\mathcal{S}) = N_{i+1}$. In other words, one can compute N_{i+1} by applying c_i to N_i instead of applying c_i to $\mathcal{S}$, thus saving time since N_i is smaller than $\mathcal{S}$.

Figure 2 illustrates the algorithm. In this figure, we assume that $\mathcal{S} = N_0 = \{x_1, x_2, ..., x_p\}$ and that the x_i are ordered so that $N = \{x_1, x_2, ..., x_k\}$ and more generally $N_i = \{x_1, x_2, ..., x_{k_i}\}$. This reordering is made possible by the inclusion relationship between the N_i sets assumption. The top part, with the horizontal arrow labeled $c = c_n$, represents the standard way of computing N, i.e. evaluate c on every element of $\mathcal{S}$, and retain only the items that satisfy c. The cost of this approach is:

$$t(c)|\mathcal{S}| = t(c)|N_0| \tag{2}$$

where $t(c)$ is the time it takes to evaluate function c on one item and $|\mathcal{S}|$ is the cardinality of $\mathcal{S}$. The rest of the figure illustrates our approach, reading from left to right. The leftmost column of the figure, labeled "$\mathcal{S}$" is an enumeration of $\mathcal{S}$. The nearest column, labeled "N_1", can be understood as follows: we evaluate c_0 on every item in $\mathcal{S}$, which yields N_1, the set of items that satisfy c_0. This is represented by the oblique arrow labeled "c_0". N_1 is enumerated vertically in this column. The cost of this step is:

$$t(c_0)|\mathcal{S}| = t(c_0)|N_0| \tag{3}$$

Reading Figure 2 from left to right illustrates that we iteratively apply c_0, c_1, ..., c_n to N_0,N_1, ..., N_n. Eventually, $c_n = c$, the target criterion, is evaluated against N_n, yielding N. The overall cost of this approach is the sum of the cost of each step:

$$\sum_{i=0}^{n} t(c_i)|N_i| \tag{4}$$

Our approach is interesting only in those situations where:

$$\sum_{i=0}^{n} t(c_i)|N_i| < t(c)|N_0| = t(c).|\mathcal{S}| \tag{5}$$

On Figure 2, the successive sets computed are represented vertically, and the successive criterion evaluations are represented horizontally. The costs can be visualized graphically if we assume that the proportions are respected, i.e. that the height of a set is proportional to its cardinality and that the width of a column is proportional to the cost of the corresponding criterion evaluation. The overall cost of our approach corresponds to the light gray surface (the upper-left "triangle"), while the cost of the standard approach is the hashed surface. With this graphical representation, it appears that if the N_i (the heights) decrease fast enough and that the $t(c_i)$ (the widths) simultaneously increase fast enough with increasing i, the light gray surface will be substantially smaller than the hashed surface. This is what we discuss in the next section.

2.2 Efficiency of the Approach

Our approach is interesting when it saves time, i.e. when equation 5 holds. This gives us a set of necessary conditions for the method to run faster than the standard approach. We will construct them recursively on n, starting with the case $n = 1$. For $n = 1$, equation 5 becomes:

$$t(c_0)|N_0| + t(c_1)|N_1| < t(c).|\mathcal{S}| \tag{6}$$

$$\Rightarrow \quad t(c_0) < t(c)\tfrac{|N_0|-|N_1|}{|N_0|} = t(c)\tfrac{|\mathcal{S}|-|N_1|}{|\mathcal{S}|} \tag{7}$$

For $n = 2$, equation 5 becomes:

$$t(c_0).|N_0| + t(c_1).|N_1| + t(c_2).|N_2| < t(c).|S| \tag{8}$$

where $c_2 = c$. If we assume that equation 7 holds, we get the sufficient condition:

$$t(c_1) < t(c)\frac{|N_1| - |N_2|}{|N_1|} \tag{9}$$

and so on. Finally, we have the following sufficient conditions for our approach to be interesting in terms of computation time:

$$t(c_i) < t(c)\frac{|N_i| - |N_{i+1}|}{|N_i|}, \forall i \in [0, n-1] \tag{10}$$

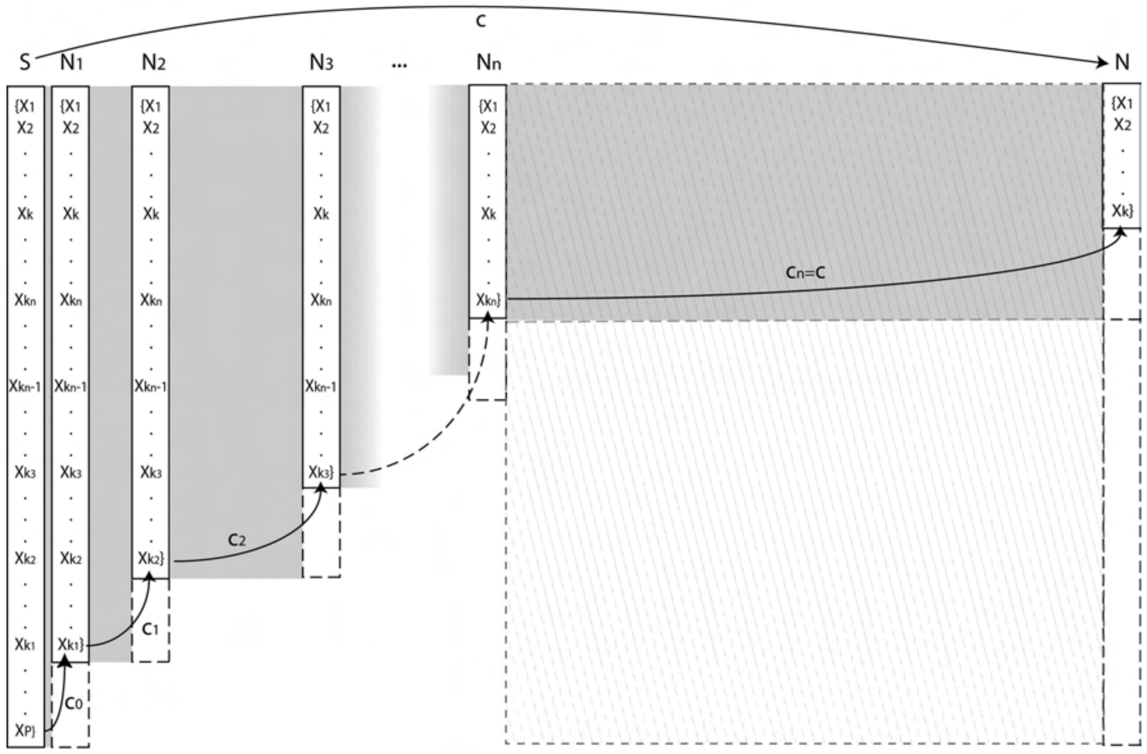
c
S
N1
N2
N3
...
Nn
N
{X1 X2 . . . Xk . . . Xkn . . . Xkn-1 . . . Xk3 . . . Xk2 . . . Xk1 . . . XP}
Cn=C
C2
C1
C0

Figure 2: Illustration of the algorithm. The cumulated cost of the successive steps appears as the light gray area, whereas the cost of the direct NN calculation appears as the stripped area.

$|N_i|$ is related to the precision with which c_i approximates the target criterion c[5]: the less precise is c_i, the larger is the smaller set of items that satisfy c_i which contains all items that satisfy c. Equation 10 thus requires that at each step i, the precision of the c_i's increases faster than their complexity.

2.3 Implementing the Approach

For a given problem, one thus needs to find a sequence of steps (the successive c_i's and N_i's) that both verifies properties P_1, P_2, and P_3 and equation 10. Equation 10 holds on the cardinalities of the successive result sets (the N_i sets). Therefore, our approach is worth applying to problem for which the cardinalities of the result sets can be computed or estimated easily. Section3 illustrates a case where the cardinalities are estimated once, even at a high cost, for a whole family of criteria.

For a given set S and a given criterion c, our approach is based on the existence of a series $(c_i)_i$ that satisfies properties P_1, P_2, and P_3. Such a series can easily be found for the class of criteria that possess a tradeoff parameter p. Let us assume that p takes value in a finite set $P = \{0, ..., n\}$ (using quantization if needed). In this context, the series $(c_i)_{i \in P}$ does not necessarily satisfy equation 10, and if it does, there may exist sub-series of $(c_i)_{i \in P}$ that allow a more efficient implementation of our approach. More precisely, given set S and criterion series $(c_i)_{i \in P}$, there exist 2^n sub-series $(c'_i)_{i \in P' \subseteq P}$ of $(c_i)_{i \in P}$, corresponding to different steps of the approach. (Note that if $(c'_i)_i$ is a sub-series of $(c_i)_i$, an item c'_j is one of the c_i with $j \leq i$, and similarly, $N'_j = N_i$.) The cost of the approach for $(c'_i)_i$ is $\sum_{i \in P'} t(c'_i)|N'_i|$. Among those sub-series, at least one of them is optimal, i.e. there is at least one for which minimizing $\sum_{i \in P'} t(c'_i)|N'_i|$. Note that when the optimal sub-series contains only the target criterion c, our approach equals the standard approach.

To implement the approach optimally, one needs to compute the optimal $(c'_i)_i$. In general, one cannot compute the cost of every 2^n sub-series. However, this can be achieved very efficiently using dynamic programming, as illustrated by the following algorithm:

```
bestSubSeries(n)
   if memValue(n)already computed
      return memValue(n)
   min $\leftarrow +\infty$
   for p $\leftarrow 0$ to $n-1$
      tmp $\leftarrow$ bestSubSeries(p)
      c $\leftarrow$ cost(tmp $\cup$ $\{n\}$)
      if c $<$ min
         result $\leftarrow$ tmp $\cup$ $\{n\}$
         min $\leftarrow$ c
      end if
   end for
   memValue(n) $\leftarrow$ result
   return result
end bestSubSeries

cost(listOfIndices)
    $\sum t(c'_i)|N'_i|$ for $i$ in listOfIndices
end cost
```

[5]In Section 3, we will show that in terms of information retrieval, $|N_i|$ is related to the precision at recall 1

3 TIMBRE SIMILARITY EXPERIMENTS

In this section, we apply the algorithm described in Section 2 to the practical task of calculating the n nearest neighbor of a song according to the timbre similarity measure presented in Aucouturier and Pachet (2004).

3.1 The Precision-Cputime Tradeoff

We sum up here the timbre similarity algorithm as presented in Aucouturier and Pachet (2004). The signal is first cut into frames. For each frame, we estimate the spectral envelope by computing a set of Mel Frequency Cepstrum Coefficients (MFCCs) (Rabiner and Juang (1993)). We then model the distribution of the MFCCs over all frames using a Gaussian Mixture Model (GMM). A GMM estimates a probability density as the weighted sum of $\mathcal{M}$ simpler Gaussian densities, called components or states of the mixture. (Bishop (1995)):

$$p(x_t) = \sum_{m=1}^{m=\mathcal{M}} \pi_m \mathcal{N}(x_t, \mu_m, \Sigma_m) \tag{11}$$

where x_t is the feature vector observed at time t, $\mathcal{N}$ is a Gaussian pdf with mean μ_m, covariance matrix Σ_m, and π_m is a mixture coefficient (also called state prior probability). The parameters of the GMM are learned with the classic E-M algorithm (Bishop (1995)).

We then compare the GMM models to match the timbre of different songs, which gives a similarity measure based on the audio content of the music. We use a Monte Carlo approximation of the Kullback-Leibler (KL) distance between each duple of models A and B. The KL-distance between 2 GMM probability distributions p_A and p_B (as defined in (11)) is defined by :

$$d(A,B) = \int p_A(x) \log \frac{p_B(x)}{p_A(x)} dx \tag{12}$$

The KL distance can thus be approximated by the empirical mean :

$$\widetilde{d(A,B)} = \frac{1}{n} \sum_{i=1}^{n} \log \frac{p_B(x_i)}{p_A(x_i)} \tag{13}$$

(where n is the number of samples x_i drawn according to p_A) by virtue of the central limit theorem :

$$\lim_{n\to\infty} \left(\frac{1}{n}\sum_{i=1}^{n} X_i - \mathcal{E}(X)\right) = \frac{1}{\sqrt{n}}\mathcal{N}(0,\sigma^2) \tag{14}$$

where X is the random variable $\log \frac{p_B(x)}{p_A(x)}$, X_i a realization of X, $\mathcal{E}(X)$ the mean of X and $\mathcal{N}(0,\sigma^2)$ a normal distribution of mean 0 and variance σ^2, the variance of X.

The precision of the approximation is clearly dependent on the number of samples n drawn from the distributions, which we call Distance Sample Rate (dsr). Figure 3 shows the influence of dsr on the precision of the measure, as defined in Aucouturier and Pachet (2004) (on a semi-logarithmic scale). We see that the DSR has a positive influence on the precision when it increases from 1 to 2000, and that further increase has little if any influence. Figure 3 also shows the (rescaled) cpu time profile, which is a linear function of dsr. It appears that first, the algorithm exhibits a precision-cputime tradeoff (using the dsr as tradeoff parameter p), and second that, for small dsr's, the precision of the measure increases faster than its cpu time, which makes it a good candidate for the NN algorithm presented in Section 2.

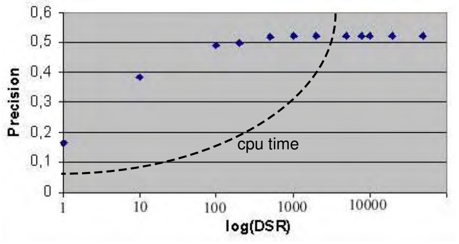

Figure 3: Influence of the distance sample rate on the precision and cpu time of the timbre similarity algorithm

3.2 Formulation of the Problem

We apply the algorithm described in Section 2 to the task of computing the 100 nearest neighbors of an arbitrary seed song in a database containing 15,554 music files, with respect to the target distance d. In our problem, d is the timbre distance described above using $dsr = 2000$, which is considered to be an ideal setting.

The distance algorithm has a tradeoff parameter $p = dsr$ which takes its integer values in $P = \{1, ..., 2000\}$, and we refer to the instances of the distance which uses p as d_p. Notably, $d = d_{2000}$. The cost of computing d_p is linear in p, and the precision of d_p increases with p.

This problem fits into the scheme presented in Section 2 if one states it as follows:

- S is the collection of music files
- d_p is the Monte Carlo approximation of the KL distance with p sampling points
- s is an element of S
- $N_p(s)$ is the set of the 100 nearest neighbors of s wrt d_p. In particular, what we want to compute is $N_{2000}(s)$, the set of the 100 nearest neighbors of s wrt $d = d_{2000}$

Given s in S, $\forall i \in \{1, ..., 2000\}$, we define the result sets $N_i \subseteq S$ as follows: $\forall i \in \{1, ..., 2000\}$, N_i is the smallest subset of S such that:

$$\forall x \in N_{2000}, \forall y \in S, d_i(x,s) \geq d_i(y,s) \Rightarrow y \in N_i \tag{15}$$

In terms of information retrieval, if we define the set of relevant documents as $N_{2000}(s)$, we can observe that

- $|N_i|$ is the number of documents retrieved by d_i when recall[6] = 1, i.e. when we have retrieved all the relevant documents.

[6]Recall is the ratio of the number of relevant documents retrieved to the total number of relevant documents in the database.

- $|N_i|$ is inversely related to the precision[7] of the measure d_i at recall 1.

$$precision(d_i) = \frac{|N_{2000}(s)|}{|N_i|} = \frac{100}{|N_i|} \quad (16)$$

We can now define c_i by:

$$c_i(x) = true \Leftrightarrow x \in N_i(s) \quad (17)$$

Let us demonstrate that properties P_1, P_2 and P_3 hold for the c_i thus defined:

- $P1$ is satisfied since the cost of computing d_p is linear in p
- $P2$ and $P3$ are satisfied statistically, since the precision of d_p increases with p and by construction of the N_i result sets.

Therefore, one can apply our approach to the problem of computing N_{2000} for seed song s.

3.3 Practical Implementation

In order to find the optimal series of $(c_i)_i$ that minimizes the total cputime of our approach for a given query on $N_{2000}(s)$, we need to estimate the $|N_i|$ for a (large) set of $i \in \{1, ..., 2000\}$. One way to estimate $|N_i|$ is to actually compute the set N_i, i.e.

- apply d_{i-1} on $N_0 = S$ in order to sort the songs in S by distance to s according to d_{i-1}
- find the maximum rank over all songs in N_{2000}. It corresponds to the rank after which all the items of $N_{2000}(s)$ have been retrieved, i.e. $|N_i|$

However, this direct approach has two major problems.

- The set of $|N_i|$ depends on the seed song, so in theory, we have to apply this procedure for each seed song before being able to find the optimal sequence of steps. This is unpractical, as estimating the $|N_i(s)|$ for a given s is itself longer than the direct calculation of $N_{2000}(s)$ with the standard approach. Moreover, it's a chicken and egg problem, as computing the $|N_i(s)|$ requires to know $N_{2000}(s)$.
- The P distances d_i are stochastic algorithms based on Monte Carlo, which never return the same distance $d_i(s,t)$ between 2 given songs s and t twice (although the variance on the results obviously decreases as dsr increases). Hence, for a given seed song s, the $|N_i(s)|$'s themselves should be averaged over several runs of the above procedure.

To overcome these limitations, we propose to estimate a unique set of $|\widetilde{N_i}|$ for the whole database, by applying the above procedure to a few random songs in the database and averaging the results. This has the drawback that the successive inclusion property (Property P_4) is only statistically verified for the estimated $|\widetilde{N_i}|$, and we have no insurance that, for a given seed song s, at a given step i, the set of items $\widetilde{N_i}$ actually contains all the items in $N_{2000}(s)$. It follows that the final set of items returned by the algorithm after a given series of steps $(c_i)_i$ is only an estimate $\widetilde{N_{2000}}(s)$ of the actual set $N_{2000}(s)$, associated with a precision

$$p((c_i)_i, s) = \frac{|\widetilde{N_{2000}}(s) \cap N_{2000}(s)|}{|N_{2000}(s)|} \quad (18)$$

[7]The precision is the ratio of the number of relevant documents retrieved to the total number of documents retrieved

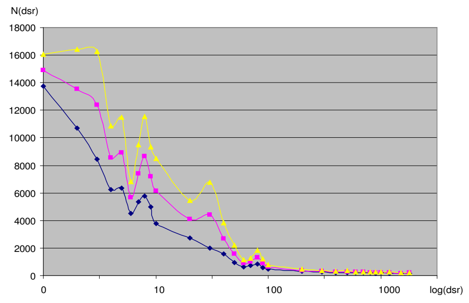

Figure 4: Convergence profile of the N_i, averaged over 50 NN timbre queries.

Figure 4 shows the estimated $|\widetilde{N_i}|$ for $i = 1, ..., 2000$, computed on the test database by averaging the $N_i(s)$ over $n_s = 50$ random songs. The darkest curve corresponds to the average $m = \frac{1}{n_s}\sum_{k=1}^{n_s} |N_i(s_k)|$, the medium curve corresponds to the sum $m + \sigma$ of the average m and the standard deviation σ of the $|N_i(s_k)|$, and the lightest curve to $m + 2\sigma$.

3.4 Results

We apply our algorithm to the task of calculating the 100 nearest neighbors of a given seed song according to the timbre similarity described above. Table 1 shows the optimal sequence of steps $(c_i)_i$ obtained with dynamic programming (see Section 2.3), and the associated cost measured by $\sum_i |N_i| t(c_i)$. We compare the results using the 3 sets of estimated $|\widetilde{N_i}|$ in Figure 4 and an additional set obtained by downsizing the $|N_i|$ by 25%. For dynamic programming, we make the assumption that the cputime is linear $t(c_i) = \alpha.i + \beta$, with $\alpha = 1$ and $\beta = 0$. It appears that the optimal sequences differ slightly whether we consider the $|\widetilde{N_i}|$ with or without standard deviation. The optimal sequence yields an algorithm which is theoretically more than 30 times faster than the standard approach.

Table 2 shows the measured performance (cputime and precision) of the actual implementation of the algorithm for the same sequences of steps. Overall, the cpu performance is very good (we achieve speed improvement factors greater than 30) while still preserving near perfect precision (we retrieve 98% of the 100 true nearest neighbors). We observe that as the $|N_i|$ increase, the precision of the results increases (we are less subjected to accidently pruning relevant nearest neighbors) but also the cpu time. We may observe that the achieved cputime rates are lower

Table 1: Optimal sequences as predicted by dynamic programming

Strategy	**Steps ($\lvert N_i \rvert$,i)**	**cost (% standard)**
standard	{15554, 2000}	31,080k (100%)
best (mean)	{15554, 6}, {4501, 20}, {2710, 60}, {652, 200}, {290, 400}, {218, 2000}	1,028k (3.3%)
best (mean - 25%)	{15554, 6}, {3375, 20}, {2032, 60}, {489, 200}, {217, 400}, {163, 2000}	793k (2.6%)
best (mean + σ)	{15554, 6}, {4090, 60}, {894, 200}, {374, 400}, {264, 2000}	1,195k (3.9%)
best (mean + 2σ)	{15554, 6}, {6819, 60}, {1136, 200}, {458, 400}, {310, 2000}	1,532k (4.9%)

than the theoretical predictions (about 1% absolute). This can be explained by the following points :

- The optimal sequence found by dynamic programming and its expected performance were computed using a very simple cpu time model $t(c_i) = i$. This doesn't include e.g. the overhead cost of file I/O (retrieving the GMMs from the database, writing the results, etc.)
- The distance algorithm was not reimplemented to support our recursive approach, i.e. the same executable is run for the successive values of dsr. While this makes the algorithm generic (no need to reprogram the distance algorithm it uses), this has an unnecessary cost: each step adds the overhead of its own system call (the executable is called from Java), initialization, file I/O (all the needed GMM files are re-opened at each step, while $|N_{i+1}| - |N_i|$ files are common between each successive call), Gaussian sampling (at each step, dsr_i points are sampled from the Gaussians, while only $dsr_{i+1} - dsr_i$ new points are needed). Most of these overhead costs are not accounted for in the theoretical predictions.

Table 2: Measured cputime and precision of several sequences of steps $(c_i)_i$

Series	**cpu-time (% stand.)**	**precision**
standard	663.75 (100%)	100%
best (mean)	27.07 (4.0%)	98.2%
best (mean - 25%)	20.98 (3.1%)	94.0%
best (mean + σ)	33.91 (5.1%)	98.9%
best (mean + 2σ)	39.19 (6.0%)	99.0%

3.5 Monte-Carlo Improvements

The previous results show that the speed of convergence of the precision of the successive approximations with increasing dsr is an important factor to ensure both an important speed improvement and a precise result set. We present here several variants of the Monte Carlo sampling meant to improve the convergence of the approximation. Note that these variants don't improve the overall precision of the distance algorithm, but rather enable faster calculation using the algorithm described in this paper.

3.5.1 Semi-Deterministic Gaussian sampling

For very small dsr (1-10), it may be appropriate to maximize the prototypicality of the drawn samples from a GMM by not drawing them at random (i.e. first drawing a Gaussian component according to the a priori distribution, and then drawing a point from the chosen Gaussian), but rather by deterministically choosing the centers of the Gaussian components. More precisely, we can use 3 strategies to sample from the GMMs.

1. successively pick the center of the largest Gaussian components by decreasing importance (first the largest component, then second largest, etc.)
2. pick the center of a randomly drawn Gaussian component (according to the a priori distribution)
3. normal sampling : random Gaussian, random point in the Gaussian.

The first strategy is more deterministic than the second, which itself is more than the third, hence we can explore the whole space of such variants using 2 cut points, cut_1 and cut_2. To sample $dsr = n$ points from a Gaussian, apply the first strategy for the first cut_1 points, then switch to the second strategy for $cut_2 - cut_1$ points, and finally use the third strategy for the remaining $n - cut_2$ points. Figure 5 shows an exploration of the space defined by (cut_1, cut_2), where cut_1 and cut_2 take values in $\{0, 1, 5, 10, 20, 50\}$. We estimate the precision convergence by computing $a = \sum_{i=1}^{2000} i.|N_i|$, which corresponds to the area of the light gray curve in Figure 2 in the case where $t(c_i)$ is linear. The smaller the a value, the faster the convergence. We compute a by averaging over 50 nearest neighbor queries on randomly drawn items in the test database. Figure 5 shows that an hybrid sampling strategy which consists in first drawing the center of the largest Gaussian ($cut_1 = 1$), then drawing the centers of 9 randomly drawn Gaussians ($cut_2 = 10$), and finish sampling with the standard strategy, is more than twice as effective than the standard strategy all through (which corresponds to $\{cut_1 = 0, cut_2 = 0\}$.

3.5.2 Antithetic variant method

The antithetic variant method is a simple improvement method of Monte Carlo's convergence, which is independent of the distribution type. It simply generates an extra random number y for every generated number x by changing its sign $y = -x$. This makes the empirical mean of the sequence tend to 0 with a significant increase in convergence. The samples are then shifted and scaled in match the target distribution's mean and variance.

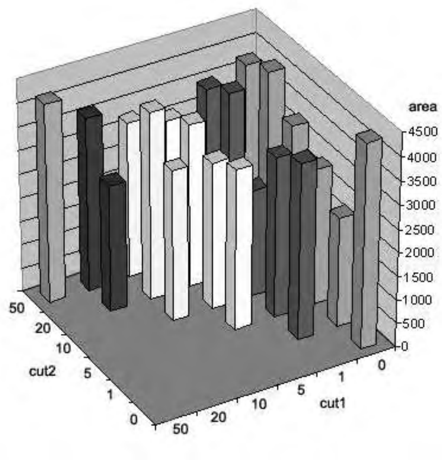

Figure 5: Exploration of the sampling variants defined by (cut_1, cut_2). The optimal is a hybrid sampling strategy with $cut_1 = 1$ and $cut_2 = 10$.

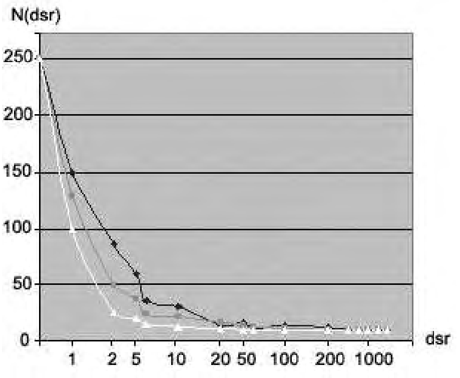

Figure 6: Comparison of sampling strategies

Figure 6 shows the decreasing $|N_i|$ in function of dsr for the normal sampling strategy (black curve), sampling with antithetic variant method (gray curve) and a combination of antithetic variant and the semi-deterministic sampling (white curve). This shows that a significant convergence increase can be achieved using these methods.

4 Conclusion

We described a non-intrusive algorithm to quickly compute the N nearest neighbors according to arbitrary similarity measures which present a tradeoff between precision and cputime. The algorithm uses successive approximations of the measure to compute more and more expensive measures on smaller and smaller sets. We achieve speed improvement factors as high as 30, while still preserving more than 98% precision, which paves the way for real-world sized music databases.

ACKNOWLEDGEMENTS

This work was partially founded by the SemanticHifi European project (http://shf.ircam.fr/).

References

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

C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford Press, 1995.

T. Bozkaya and M. Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems*, pages 1–34, 1999.

P. Ciacca, M. Patella, and P. Zezula. M-tree: an efficient access method for similarity search in metrix spaces. In *Proceedings of the 23rd International Conference on Very Large Databases*, 1997.

J. T. Foote. Content-based retrieval of music and audio. In *Multimedia Storage and Archiving Systems II, Proc. of SPIE*, 1997. pages 138-147.

A. Juan, E. Vidal, and P. Aibar. Fast k-nearest-neighbours searching through extended versions ofthe approximating and eliminating search algorithm (aesa). In *Proceedings of the Fourteenth INternational Conference on Pattern Recognition*, Brisbane, Australia, August 1998.

D. Miranker, W. Xu, and R. Mao. Mobios: a metric-space dbms to support biological discovery. In *Proceedings Of the International Conference On Scientific and Statistical Database Management Systems (SSDBM)*, 2003.

E. Pampalk, S. Dixon, and G. Widmer. On the evaluation of perceptual similarity measures for music. In *Proceedings of the Sixth International Conference on Digital Audio Effects DAFX, London (UK)*, September 2003.

L. R. Rabiner and B. H. Juang. *Fundamentals of speech recognition*. Prentice-Hall, 1993.

J. Reiss, J.-J. Aucouturier, and M. Sandler. Efficient multidimensional searching routines for mir. In *Proceedings of the 2nd International Symposium on Music Information Retrieval (ISMIR), Bloomington, Indiana (USA)*, October 2001.

H. Samet. *Applications of Spatial Data Structures*. Addison-Wesley, 1989.

R. Typke, P. Giannopoulos, R. C. Veltkamp, F. Wiering, and R. van Oostrum. Using transportation distances for measuring melodic similarity. In *Proceedings of the International Conference on Music Information Retrieval*, October 2003.

E. Vidal, F. Casacuberta, J.M. Benedi, J. Lloret, and H. Rulot. On the verification of triangle inequality by dynamic time-warping dissimilarity measures. *Speech Communication*, 7(1):67–79, 1988.

M. Welsh, N. Borisov, J. Hill, R. von Behren, and A. Woo. Querying large collections of music for similarity. Technical Report Technical Report UCB/CSD00 -1096, U.C. Berkeley Computer Science Division, 1999.

E. Wold and T. Blum. Content based classification, search and retrieval of audio. *IEEE Multimedia*, 3(3), 1996.