Using point-set compression to classify folk songs

Meredith, David

Published in:
Proceedings of the Fourth International Workshop on Folk Music Analysis (FMA2014)

Publication date:
2014

Document Version
Early version, also known as pre-print

Link to publication from Aalborg University

Citation for published version (APA):
USING POINT-SET COMPRESSION TO CLASSIFY FOLK SONGS

David Meredith
Aalborg University
dave@create.aau.dk

ABSTRACT

Thirteen different compression algorithms were used to calculate the normalized compression distances (NCDs) between pairs of tunes in the Annotated Corpus of 360 Dutch folk songs from the collection *Onder de groene linde*. These NCDs were then used in conjunction with the 1-nearest-neighbour algorithm and leave-one-out cross-validation to classify the 360 melodies into tune families. The classifications produced by the algorithms were compared with a ground-truth classification prepared by expert musicologists. Twelve of the thirteen compressors used in the experiment were based on the discovery of translational equivalence classes (TECs) of maximal translatable patterns (MTPs) in point-set representations of the melodies. The twelve algorithms consisted of four variants of each of three basic algorithms, COSIATEC, SIATECCOMPRESS and Forth's algorithm. The main difference between these algorithms is that COSIATEC strictly partitions the input point set into TEC covered sets, whereas the TEC covered sets in the output of SIATECCOMPRESS and Forth’s algorithm may share points. The general-purpose compressor, bzip2, was used as a baseline against which the point-set compression algorithms were compared. The highest classification success rate of 77–84% was achieved by COSIATEC, followed by 60–64% for Forth’s algorithm and then 52–58% for SIATECCOMPRESS. When the NCDs were calculated using bzip2, the success rate was only 12.5%. The results demonstrate that the effectiveness of NCD for measuring similarity between folk-songs for classification purposes is highly dependent upon the actual compressor chosen. Furthermore, it seems that compressors based on finding maximal repeated patterns in point-set representations of music show more promise for NCD-based music classification than general-purpose compressors designed for compressing text strings.

1. INTRODUCTION

For over a century, musicologists have been interested in measuring similarity between folk song melodies (Scheurleer, 1900; van Kranenburg et al., 2013), primarily with the purpose of classifying such melodies into families (Bayard, 1950) of tunes that have a common ancestor in the tree of oral transmission. Researchers have used a plethora of different features and methods in their attempts to automate (or at least formalize) this process of folk-song classification (see van Kranenburg et al., 2013, for an overview). In some cases, such methods have led to almost perfect models of the classifications produced by expert musicologists. For example, van Kranenburg et al. (2013) report a 99% success rate for classifying a set of 360 Dutch folk songs with a method based on local-features and string alignment. In contrast, in the study reported here, a universal, generally-applicable similarity metric, normalized compression distance (NCD, Li et al., 2004), is used to classify folk-song melodies based on compressing the melodies by discovering maximal repeated patterns within them.

Normalized compression distance has been used in several music classification studies in the past (Cilibrasi et al., 2004; Li & Sleep, 2004, 2005; Hillewaere et al., 2012). However, in these studies, only general-purpose compressors such as those based on the Lempel-Ziv algorithm (Ziv & Lempel, 1977, 1978) and bzip2 (Seward, 2010) have been used. In the study reported here, NCD was used to classify folk songs using a number of different compression algorithms specifically designed for producing compact structural analyses of pieces of music from symbolic encodings in the form of point sets (Meredith et al., 2003; Meredith, 2006a; Forth & Wiggins, 2009; Forth, 2012; Meredith, 2013). The results suggest that the choice of compressor has a very marked effect on the classification success rate.

2. NORMALIZED COMPRESSION DISTANCE

Li et al. (2004) introduced the normalized information distance (NID), a universal similarity metric based on Kolmogorov complexity (Li & Vitányi, 2008). The Kolmogorov complexity of any object is the length in bits of the shortest program that generates the object as its only output. The NID defines the distance between any two objects, x and y, as

\[
d(x, y) = \frac{\max\{K(x | y^r), K(y | x^r)\}}{\max\{K(x), K(y)\}}
\]

where \(K(x)\) is the Kolmogorov complexity of x and \(K(x | y^r)\) is the conditional complexity of x given a description of y whose length is equal to the Kolmogorov complexity of y. The Kolmogorov complexity of an object, however, is not computable. Therefore, \(K(x)\) has to be substituted in practice by the length of a compressed encoding of x generated using a real-world compressor. Li et al. (2004) therefore propose the normalized compression distance (NCD) as a practical alternative to the NID and define it as follows:

\[
\text{NCD}(x, y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}}
\]

where \(C(x)\) is the length of a compressed encoding of x and \(C(xy)\) is the length of a compressed encoding of a concatenation of x and y.
In the dataset, \( v \) vector, there exists a vector, \( v \) said to be translatable by the vector \( v \) translated by \( v \) equivalent to the pattern of crosses. A pattern, \( P \) graphs in Figure 2, the pattern of circles is translationally equivalent to the pattern of crosses. Figure 1 shows an example of such a dataset. When the music to be analysed is modal or uses the major-minor tonal system, the output of the algorithms described below is typically better when morphetic pitch is used. If morphetic pitch information is not available (e.g., because the data is only available in MIDI format), then, for modal or tonal music, it can be reliably computed from a representation that provides the chromatic pitch (i.e., MIDI note number) of each note, by using an algorithm such as PS13s1 (Meredith, 2006b, 2007). For pieces of music not based on the modal or major-minor tonal system, using chromatic pitch may give better results than using morphetic pitch.

3. REPRESENTING MUSIC WITH POINT SETS

In the algorithms considered in this paper, it is assumed that the music to be analysed is represented in the form of a multi-dimensional point set called a dataset, as described by Meredith et al. (2002). All the algorithms described below work with datasets of any dimensionality. However, it will be assumed here that each dataset is a set of two-dimensional points, \( (t, p) \), on an integer lattice, where \( t \) and \( p \) are, respectively, the onset time in tatum and the chromatic or morphetic pitch (Meredith, 2006b, 2007; Meredith et al., 2002) of a note or sequence of tied notes in a score. Figure 1 shows an example of such a dataset. When the music to be analysed is modal or uses the major-minor tonal system, the output of the algorithms described below is typically better when morphetic pitch is used. If morphetic pitch information is not available (e.g., because the data is only available in MIDI format), then, for modal or tonal music, it can be reliably computed from a representation that provides the chromatic pitch (i.e., MIDI note number) of each note, by using an algorithm such as PS13s1 (Meredith, 2006b, 2007). For pieces of music not based on the modal or major-minor tonal system, using chromatic pitch may give better results than using morphetic pitch.

4. MAXIMAL TRANSLATABLE PATTERNS

If \( D \) is a dataset, then any subset of \( D \) may be called a pattern. If \( P_1, P_2 \subseteq D \), then \( P_1, P_2 \), are said to be translationally equivalent, denoted by \( P_1 \equiv_T P_2 \), if and only if there exists a vector \( v \), such that \( P_1 \) translated by \( v \) is equal to \( P_2 \). That is,

\[
P_1 \equiv_T P_2 \iff (\exists v \mid P_2 = P_1 + v), \tag{1}
\]

where \( P_1 + v \) denotes the pattern that results when \( P_1 \) is translated by the vector \( v \). For example, in each of the graphs in Figure 2, the pattern of circles is translationally equivalent to the pattern of crosses. A pattern, \( P \subseteq D \), is said to be translatable within a dataset, \( D \), if and only if there exists a vector, \( v \), such that \( P + v \subseteq D \). Given a vector, \( v \), then the maximal translatable pattern (MTP) for \( v \) in the dataset, \( D \), is defined and denoted as follows:

\[
\text{MTP}(v, D) = \{ p \mid p \in D \land p + v \in D \} \tag{2}
\]

5. TRANSLATIONAL EQUIVALENCE CLASSES

When analysing a piece of music, we typically want to find all the occurrences of an interesting pattern, not just one occurrence. Thus, if we believe that MTPs are related in some way to the patterns that listeners and analysts find interesting, then we want to be able to find all the occurrences of each MTP. Given a pattern, \( P \), in a dataset, \( D \), the translational equivalence class (TEC) of \( P \) in \( D \) is defined and denoted as follows:

\[
\text{TEC}(P, D) = \{ Q \mid Q \equiv_T P \land Q \subseteq D \}. \tag{3}
\]

That is, the TEC of a pattern, \( P \), in a dataset contains all and only those patterns in the dataset that are translationally equivalent to \( P \). Figure 2 shows some examples of maximal translatable patterns.

We define the covered set of a TEC, \( T \), denoted by \( \text{COV}(T) \), to be the union of the patterns in the TEC, \( T \). That is,

\[
\text{COV}(T) = \bigcup_{P \in T} P. \tag{4}
\]

Here, we will be particularly concerned with MTP TECs—that is, the translational equivalence classes of the maximal
A TEC, $T = \text{TEC}(P, D)$, contains all the patterns in the dataset, $D$, that are translationally equivalent to the pattern, $P$. Suppose $T$ contains $n$ translationally equivalent occurrences of the pattern, $P$, and that $P$ contains $m$ points. There are at least two ways in which one can specify $T$. First, one can explicitly specify each of the $n$ patterns in $T$ by listing all of the $m$ points in each pattern. This requires one to write down $mn$, $k$-dimensional points or $kmn$ numbers. Alternatively, one can explicitly list the $m$ points in just one of the patterns in $T$ (e.g., $P$) and then give the $n-1$ vectors required to translate this pattern onto its other occurrences in the dataset. This requires one to write down $m$, $k$-dimensional points and $n-1$, $k$-dimensional vectors—that is, $k(m+n-1)$ integers. If $n$ and $m$ are both greater than one, then $k(m+n-1)$ is less than $kmn$, implying that the second method of specifying a TEC gives us a compressed encoding of the TEC. Thus, if a dataset contains at least two occurrences of a pattern containing at least two points, it will be possible to encode the dataset in a compact manner by representing it as the union of the covered sets of a set of TECs, where each TEC, $T$, is encoded as an ordered pair, $(P, V)$, where $P$ is a pattern in the dataset, and $V$ is the set of vectors that translate $P$ onto its other occurrences in the dataset. When a TEC, $T = (P, V)$, is represented in this way, we call $V$ the set of transalters for the TEC and $P$ the TEC's pattern. We also denote and define the compression ratio of a TEC, $T = (P, V)$ as follows:

$$\text{CR}(T) = \frac{\text{COV}(T)}{|P| + |V|}.$$  \hspace{1cm} (5)

In this paper, the pattern, $P$, of a TEC used to encode it as a $(P, V)$ pair will be assumed to be the lexicographically earliest occurring member of the TEC (i.e., the one that contains the lexicographically least point).

6. THE ALGORITHMS

6.1 SIA

All of the compression algorithms considered in this paper are based on Meredith, Lemström and Wiggins' SIA algorithm (Meredith et al., 2001, 2002; Meredith, 2006a). SIA finds all the maximal translatable patterns in a set of $n$, $k$-dimensional points in $\Theta(kn^2 \log n)$ time and $\Theta(kn^2)$ space. Figure 4 describes how the algorithm works with a simple example and Figure 5 gives pseudocode for a straight-forward implementation of SIA. In the pseudocode used in this paper, unordered sets are denoted by italic upper-case letters (e.g., $D$ in Figure 5). Ordered sets are denoted by boldface upper-case letters (e.g., $V$, $D$ and $M$ in Figure 5). When written out in full, ordered sets are denoted by angle brackets, “⟨⟩”. Concatenation is denoted by “⟨⟩” and the assignment operator is “←”. $A[i]$ denotes the $(i+1)$th element of the ordered set (or one-dimensional array), $A$. (i.e., zero-based indexing is used). If $B$ is an ordered set of ordered sets (or a two-dimensional array), then $B[i][j]$ denotes the $(j+1)$th element in the $(i+1)$th element of $B$. Elements in arrays of higher dimension are indexed analogously. Block structure is indicated by indentation alone.

The algorithm can easily be modified so that it only generates MTPs whose sizes lie within a particular user-specified range. It is also possible for the same pattern to be the MTP for more than one vector. If this is the case, there will be two or more (pattern, vector) pairs in the output of SIA that have the same pattern. This can be avoided and the output can be made more compact by generating instead a list of (pattern, vector set) pairs, such that the vector set in each pair contains all the vectors for which the pattern is an MTP. In order to accomplish this, we merge the vectors for which a given pattern is the MTP into a single vector set which is then paired with the pattern in the output.
strictly partitions a dataset, \( T \) in an MTP, within \( T \), such that \( D = \bigcup_{T \in T} \text{COV}(T) \) and \( \text{COV}(T_1) \cap \text{COV}(T_2) = \emptyset \) for all \( T_1, T_2 \in T \) where \( T_1 \neq T_2 \). In other words, COSIATEC strictly partitions a dataset, \( D \), into the covered sets of a set of MTP TECs. If each of these MTP TECs is represented as a \( \langle \text{pattern}, \text{translator set} \rangle \) pair, then this description of the dataset as a set of TECs is typically shorter than an \( \text{in extenso} \) description in which the points in the dataset are listed explicitly.

Figure 7 shows pseudocode for COSIATEC. The first step in the algorithm is to make a copy of the input dataset which is stored in the variable \( D' \) (line 1). Then, on each iteration of the \textbf{while} loop (lines 3–6), the algorithm finds the “best” MTP TEC in \( D' \), stores this in \( T \) and adds \( T \) to \( D' \). It then removes the set of points covered by \( T \) from \( D' \) (line 6). When \( D' \) is empty, the algorithm terminates, returning the list of MTP TECs, \( T \). The sum of the number of translators and the number of points in this output encoding is never more than the number of points in the input dataset and can be much less than this, if there are many repeated patterns in the input dataset.

The \text{T}\text{GETBESTTEC} function, called in line 4 of \text{T}\text{COSIATEC}, computes the “best” TEC in \( D' \) by first finding all the MTPs using SIA, then iterating over these MTPs, finding the TEC for each MTP, and storing it if it is the best TEC so far. In this process, a TEC is considered “better” than another if it has a higher compression ratio, as defined in Eq. 5. If two TECs have the same compression ratio, then the better TEC is considered to be the one that has the higher \text{bounding-box compactness} (Meredith et al., 2002), defined as the ratio of the number of points in the TEC’s pattern to the number of dataset points in the bounding box of this pattern. Collins et al. (2011) have provided empirical evidence that the compression ratio and compactness of a TEC are important factors in determining its perceived “importance” or “noticeability”. If two distinct TECs have the same compression ratio and compactness, then, in COSIATEC, the TEC with the larger covered set is considered superior.

6.2 \text{SIATEC}

SIATEC (Meredith et al., 2001, 2002, 2003; Meredith, 2006a) computes all the MTP TECs in a \( k \)-dimensional dataset of size \( n \) in \( O(kn^2) \) time and \( O(kn^2) \) space. In order to find the MTPs, the SIA algorithm only needs to compute the vectors from each point in a dataset to each lexicographically later point. However, to compute \text{all occurrences} of the MTPs, it turns out to be beneficial in the SIATEC algorithm to compute the vectors between \text{all pairs of points}, resulting in a vector table like the one shown in Figure 6. The SIATEC algorithm first finds all the MTPs using SIA. It then uses the vector table to find all the vectors by which each MTP is translatable within the dataset. The set of vectors by which a given pattern is translatable is equal to the intersection of the columns in the vector table headed by the points in the pattern (see Figure 6). In a vector table computed by SIATEC, each row descends lexicographically from left to right and each column increases lexicographically from top to bottom. SIATEC exploits these properties of the vector table to more efficiently find all the occurrences of each MTP (Meredith et al., 2002, pp. 335–338).

6.3 \text{COSIATEC}

COSIATEC (Meredith et al., 2003; Meredith, 2006a, 2013) is a greedy point-set compression algorithm, based on SIATEC. COSIATEC takes a dataset, \( D \), as input and computes a compressed encoding of \( D \) in the form of an ordered set of MTP TECs, \( T \), such that \( D = \bigcup_{T \in T} \text{COV}(T) \) and \( \text{COV}(T_1) \cap \text{COV}(T_2) = \emptyset \) for all \( T_1, T_2 \in T \) where \( T_1 \neq T_2 \). In other words, COSIATEC strictly partitions a dataset, \( D \), into the covered sets of a set of MTP TECs. If each of these MTP TECs is represented as a \( \langle \text{pattern}, \text{translator set} \rangle \) pair, then this description of the dataset as a set of TECs is typically shorter than an \( \text{in extenso} \) description in which the points in the dataset are listed explicitly.

Figure 7 shows pseudocode for COSIATEC. The first
Collins et al. (2010, p. 6) claim that “the larger the dataset, the more likely it is that the problem will occur” and that it could prevent the SIA-based algorithms from “discovering some translational patterns that a music analyst considers noticeable or important”. Collins et al. propose that this problem can be solved by taking each MTP computed by SIA (sorted into lexicographical order) and ‘trawling’ inside this MTP “from beginning to end, returning subsets that have a compactness greater than some threshold a and that contain at least b points” (Collins et al., 2010, p. 6). This method is implemented in an algorithm that they call SIACT, which first runs SIA on the dataset and then carries out ‘compactness trawling’ (hence “SIACT”) on each of the MTPs found by SIA.

6.6 SIAR

In an attempt to improve on the precision and running time of SIA, Collins (2011, pp. 282–283) defines an SIA-based algorithm called SIAR. Instead of computing the whole region below the leading diagonal in the vector table for a dataset (as in Figure 4(b)), SIAR only computes the first r subdiagonals of this table. This is approximately equivalent to running SIA with a sliding window of size r (Collins et al., 2010; Collins, 2011).

6.7 SIATECCompress

COSIATEC uses SIA on each iteration of its while loop to compute the best TEC to add to the output encoding. Since SIA has worst-case running time $O(n^3)$ where n is the number of points in the input dataset, running COSIATEC on large datasets can be time-consuming. On the other hand, because COSIATEC strictly partitions the dataset into non-overlapping MTP TEC covered sets, it tends to achieve high compression ratios for many point-set representations of musical pieces (typically between 2 and 4 for a piece of classical or baroque music).

Like COSIATEC, the SIATECCompress algorithm shown in Figure 8 is a greedy compression algorithm based on SIATEC that computes an encoding of a dataset in the form of a union of TEC covered sets. Like Forth’s algorithm (but unlike COSIATEC), SIATECCompress runs SIATEC only once (line 1) to get a list of TECs. This list is then sorted into decreasing order of quality (line 2), where the decision as to which of any two TECs is superior is made in the same way as in COSIATEC (described above). The algorithm then finds a compact encoding, $E$, of the dataset in the form of a set of TECs. It does this by iterating over the sorted list of TECs (lines 5–12), adding a new TEC, $T$, to $E$ if the number of new points covered by $T$ is greater than the size of its (pattern, translator set) representation (lines 8–12). Each time a TEC, $T$, is added to $E$, its covered set is added to the set $D'$, which therefore maintains the set of points covered so far after each iteration. When $D'$ is equal to $D$ or all the TECs have been scanned, the for loop terminates. Any remaining uncovered points are aggregated into a residual point set, $R$, (line 13) which is re-expressed as a TEC with an empty translator set (line 15) that is added to the encoding. SIATECCompress does not generally produce as compact an encoding as COSIATEC, since the TECs in its output may share points. However, it is faster than COSIATEC and can therefore be used practically on much larger datasets. Unlike Forth’s algorithm, SIATECCompress always produces a complete cover of the input dataset.

7. USING THE ALGORITHMS TO CLASSIFY FOLK SONGS

COSIATEC, Forth’s algorithm and SIATECCompress were used to classify the melodies in the Annotated Corpus (van Krabbe et al., 2013; Volk & van Kranenburg, 2012) of 360 Dutch folk songs from the collection, Order de groene linde (Grijp 2008), hosted by the Meertens Institute and accessible through the website of the Dutch Song Database (http://www.liederenbank.nl). The algorithms were used as compressors to calculate the normalized compression distance between each pair of melodies in the collection. Each melody was then classified using the 1-nearest-neighbour algorithm with leave-one-out cross-validation. The classifications obtained were compared with a ground-truth classification of the melodies carried out by expert musicologists.

Four versions of each of the three algorithms were tested:

- the basic algorithm as described above,
- a version incorporating the compactness trawler from Collins et al.’s SIACT algorithm,
- a version using SIAR instead of SIA and
- a version using both SIAR and the compactness trawler.

As a baseline, one of the best general-purpose compression algorithms, bzip2 (Seward, 2010), was also used to calculate NCDs between the melodies.

Table 1 shows the results obtained in this task. In this table, algorithms with names containing “R” employed the SIAR algorithm with $r = 3$ in place of SIA. Algorithms
Table 1: Results of using different compressors to classify the *Annotated Corpus* of Dutch folk songs using NCD, 1-NN and leave-one-out-cross-validation. *SR* is the classification success rate, *CR* is the average compression ratio over the melodies in the *Annotated Corpus*. *CR* is the average compression ratio over the pairs of files used to obtain the NCD values.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th><em>SR</em></th>
<th><em>CR</em></th>
<th><em>CR</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>COSIATEC</td>
<td>0.8389</td>
<td>1.5791</td>
<td>1.6670</td>
</tr>
<tr>
<td>COSIARTEC</td>
<td>0.8361</td>
<td>1.5726</td>
<td>1.6569</td>
</tr>
<tr>
<td>COSIARCTTEC</td>
<td>0.7917</td>
<td>1.4547</td>
<td>1.5135</td>
</tr>
<tr>
<td>COSIACTTEC</td>
<td>0.7694</td>
<td>1.4556</td>
<td>1.5138</td>
</tr>
<tr>
<td>ForthCT</td>
<td>0.6417</td>
<td>1.1861</td>
<td>1.2428</td>
</tr>
<tr>
<td>ForthRCT</td>
<td>0.6417</td>
<td>1.1861</td>
<td>1.2428</td>
</tr>
<tr>
<td>Forth</td>
<td>0.6111</td>
<td>1.2643</td>
<td>1.2663</td>
</tr>
<tr>
<td>ForthR</td>
<td>0.6028</td>
<td>1.2555</td>
<td>1.2655</td>
</tr>
<tr>
<td>SIARCTTECompactress</td>
<td>0.5750</td>
<td>1.3213</td>
<td>1.3389</td>
</tr>
<tr>
<td>SIATECCompactress</td>
<td>0.5694</td>
<td>1.3360</td>
<td>1.3256</td>
</tr>
<tr>
<td>SIATCHTECompactress</td>
<td>0.5250</td>
<td>1.3197</td>
<td>1.3381</td>
</tr>
<tr>
<td>SIARTECompactress</td>
<td>0.5222</td>
<td>1.3283</td>
<td>1.3216</td>
</tr>
<tr>
<td>bzip2</td>
<td>0.1250</td>
<td>2.7678</td>
<td>3.5061</td>
</tr>
</tbody>
</table>

The performance of Forth’s algorithm on this task was improved by incorporating compactness trawling: using SIAR instead of SIA in Forth’s algorithm slightly reduced the performance of the basic algorithm and had no effect when compactness trawling was used. The results obtained using bzip2 were much poorer than those obtained using the SIA-based algorithms, suggesting that general-purpose compressors may fail to capture certain musical structure that is important for this task—at least when run on point-set representations of the type used in this study. Of the SIA-based algorithms, COSIATEC achieved the best compression on average, followed by SIATECCompactress and then Forth’s algorithm. COSIATEC also achieved the best success rate. However, since Forth’s algorithm performed slightly better than SIATECCompactress, it seems that degree of compression alone was not a reliable indicator of classification accuracy on this task—indeed, the best compressor, bzip2, produced the worst classifier. None of the algorithms achieved a success rate as high as the 99% obtained by van Kranenburg et al. (2013) on this corpus using several local features and an alignment-based approach. The success rate achieved by COSIATEC is within the 83–86% accuracy range obtained by Velarde et al. (2013, p. 336) on this database using a wavelet-based representation, with similarity measured using Euclidean or city-block distance.

8. CONCLUSIONS

The results in Table 1 suggest that the implicit and explicit knowledge and cognitive processes used by the musicologists who developed the ground-truth classification for the *Annotated Corpus* of Dutch folk songs can be modelled reasonably well by using normalized compression distance (NCD) as a measure of similarity. However, the results also show that the success of such an NCD-based model depends critically on which compressor one uses to produce NCDs and how encoding length is measured. In particular, in this study, compressors based on point-set pattern discovery and TEC compression-ratio performed much better than a baseline general-purpose, string-based compressor of the type used in previous studies that have used NCD for music classification.

9. ACKNOWLEDGEMENTS

The author is grateful to Peter van Kranenburg for providing the data files for the *Annotated Corpus* from the collection *Onder de groene linde* (Grijp, 2008) currently hosted by the Meertens Institute and accessible through the website of the Dutch Song Database (Nederlandse Liederenbank, www.liederenbank.nl). The work was carried out within the EU project, “Learning to Create” (Lrn2Cre8). The project Lrn2Cre8 acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET grant number 610859.


