

7 Evolutionary morphing algorithm

After probabilistic morphing, the focus of research turned to evolutionary morphing, which was explored through the development of the *TraSe* (*Transform Select*) algorithm. The hypothesis behind *TraSe* is that effective morphing can result from a series of compositional transformations applied to the source. *TraSe* allows greater control over stylistic elements of the music than the previous morphing algorithms, through user-defined weighting of numerous compositional transformations. Informal evaluation of *TraSe* involved subjective analysis of a range of audible examples. The formal evaluation of *TraSe* was mostly qualitative and occurred in the controlled environment of an online questionnaire. *TraSe* deviates substantially from any other approach in the field of compositional morphing. As a morphing technique, it directly fulfils the research objective to develop new techniques for automated and interactive compositional morphing. The music generated by *TraSe* was a substantial improvement on the parametric and probabilistic algorithms, in some cases being seen as more creative than a human composed benchmark.

As an evolutionary approach, *TraSe* is related to previous algorithmic composition systems such as GenJam (Biles 2002), Vox Populi (Moroni, Manzolli, Van Zuben and Gudwin 1990) and others (Marques, Oliveira, Vieira, and Rosa 2000; Towsey, Brown, Wright and Diederick 2001; Miranda and Biles 2007). A bibliography of evolutionary computer music is maintained by Al Biles (2007). *TraSe* is distinguished from other evolutionary music systems by features that are designed specifically for morphing.

In terms of modelling music composition and creativity, the evolutionary approach is clearly a **trial** approach to composition. This is evidenced by the generation of a large number of candidates in a selection pool, from which only one is selected according to specific criteria, in this case, the similarity with the target note sequence. This can be compared with the **abstract** approach of parametric morphing and the **heuristic** approach of probabilistic morphing.

An overview of the *TraSe* morph algorithm is provided in section 7.1. More detail on how the algorithm transforms and selects the musical sequences is outlined in 7.2. Section 7.3 specifies each of the nine different compositional transformations that are used to transform the source music. Both informal and formal evaluations of the music are provided in 7.4. The informal evaluation highlights various features of the algorithm through the analysis of musical examples. The limits of the algorithm are tested with many randomly generated source and target samples. Subsequently, the formal evaluation is presented in the form of a comprehensive critical listening questionnaire that was developed and applied to assess the musical coherence and stylistic qualities of the morphs. The questionnaire found that music composed by a human was usually

more acceptable, however, there were a number of notable exceptions. The morphs were almost unanimously said to be applicable to electronic dance and computer game music. The results also challenged the feasibility of morphing outright; in particular, the fundamental notion that morphs should be ‘smooth’ or ‘continuous’.

TraSe has by far the highest time complexity compared to the other algorithms at $O(n^3)$, due to the exhaustive searching. Because of this computation load, the morphs need to be rendered prior to realtime operation. Although realtime interactivity in playback is possible through control of the morph index, changes to the source and target music do not influence the morph in realtime.

A number of possible extensions to the current evolutionary morphing algorithm have been identified and are discussed in 7.7. Notably, algorithm designs that would significantly reduce the complexity from $O(n^3)$ to $O(n \log n)$ were comprehensively scoped out, despite not having been implemented. This means the algorithm has definite potential for true realtime operation in the future.

Overall, *TraSe* is the most novel and effective compositional morphing algorithm developed through this study. Despite this, there is still much to be done before it can consistently compare favourably to a human composer, particularly in terms of coherence. The final results were constructive and produced a number of ideas for future research.

7.1 Overview

TraSe takes two note sequences, source, S , and target, T , and produces morphed material, M , which is a hybrid transition between the two. The morph, M is a list of note sequence loops that I call **frames**, the first and last of which is S and T respectively. If we let n be the length of the frame list, $M_1 = S$ and $M_n = T$. The frames of the middle M_2, M_3, \dots, M_{n-1} constitute a sequential progression.

During playback, the morph index, Ω , determines which frame in the series is playing. Letting i be the index of any frame, ranging from $1 \leq i \leq n$, and noting that Ω is normalized between 0 and 1, M_i will hold the note sequence that is looped when $\Omega = i/n$. Logically, the morph will be **smooth** if each frame is somehow made similar to its neighbours. Smoothness is the quality of continuity, whereby each segment is perceived as being more related to the points directly ahead and behind than the others which are further away.

The musical representation includes the loop length, note onsets, durations, dynamics, scale

degrees/passing notes, scale and key; as per the *Degree Passing note (DePa)* scheme described in Chapter Four. There are two separate *TraSe* algorithms that operate in parallel – one for key and scale and another for note-level data (note onset, duration, dynamic and scale degree/passing note). I refer to the former as a **key/scale morph** and the latter as a **note morph**. Both the note morph and key/scale morph have a separate M which can be of different lengths. Unless stated otherwise, the *TraSe* processes described in this chapter are for the note morph.

During playback, the current frame of the key/scale morph and the note morph is combined to produce output with a standard note representation of pitch, duration, onset and dynamic. Having the key/scale and note morphs separate affords more specific control over the tonal elements of the morph. Early versions of *TraSe* that did not separate key/scale were more difficult to use.

If S and T are different lengths, each frame will have a length equal to the lowest common multiple. For example, if S was 6 beats and T was 4, the loops in each frame of M would be 12 beats long. Requiring the frames to be the same length is practical; a uniform length affords automated comparisons between sequences, due to their notes occupying an identical space. It also affords the combination of material.

Recall that i is an index to M . *TraSe* sequentially fills each frame of M , starting with $i = 2$, and finishing with $i = n$. Each frame, M_i , is created by passing the previous frame, M_{i-1} , through a chain of compositional transformation functions: let this be the list C . The functions in the chain C perform musical transformations that a composer might attempt, for example *harmonise* creates or removes harmonies at particular intervals, while *rate* speeds up or slows down the music by particular commonly used ratios, looping when necessary.

To generate the frame M_i , the first transformation, C_1 , is passed the previous frame, M_{i-1} . The second transformation is passed the output of the first, and this continues until the last transformation. The last transformation is fed the output of the second last transformation to create M_i . Letting m equal the length of C , or $m = |C|$, this means: $M_i = C_m(C_{m-1}(C_{m-2}(\dots(C_1(M_{i-1}))))$. *TraSe* stops generating frames when the current frame is judged as being similar enough to the target, T , by a variable for ‘cut-off’ that is specified by the user. Each frame is part of a sequence of frames that is the morph: $M_1, \dots, M_i, \dots, M_n$.

The evolutionary component of *TraSe* is in the fact that each transformation produces a range of different sequences from which only one is selected, through a comparison with the target sequence. This is essentially a Genetic Algorithm (GA), as the transformations are ‘mutations’

and the comparison to the target is a ‘fitness function’. This transform-select process is quite complex and will be explained in more detail in the following section 7.2. Allowing specific compositional transformations and dissimilarity measures (see 7.3) to be designed and plugged into the algorithm enables elements of compositional style to be specified. For now, a simple high-level overview of the *TraSe* process has been summarised in pseudocode:

```
// inputs:
S      is the source note sequence
T      is the target note sequence
C      is a list of compositional transformations that provides data and functions used
        in the TRANSFORM-SELECT process.
cutoff defines how similar the current frame needs to be to the T to end the process
// functions:
MAKE-SAME-LENGTH
    Loops the inputs until they are as long as the lowest common multiple of their
    lengths so that they are the same length
FIND-DISSIMILARITY
    Rates the dissimilarity of two inputs, from the same (0) to most dissimilar (1)
TRANSFORM-SELECT
    Applies a transformation and selects an output, more similar to T than the input.
// output:
M      List of note sequences where the first item is the same as S, subsequent patterns
        are more and more similar to T and the last item is the same as T
function TRASe (note sequence S, note sequence T, transformation list C)
    returns note sequence list M {
        MAKE-SAME-LENGTH(S, T)
        i = 0; // index of current frame in M
        M[i] = COPY(S)
        WHILE( FIND-DISSIMILARITY(M[i],T) < cutoff ){
            i = i + 1 // increment index to the next frame
            M[i] = COPY(M[i-1])
            FOR(j = 0; j < LENGTH(C); j++) {
                M[i] = TRANSFORM-SELECT(C[j], M[i], S, T, i);
            }
        }
        return M
    }
```

Figure 1 Pseudocode overview of *TraSe*.

7.2 Transforming and selecting

Each transformation has a set of possible parameter configurations. A pool of candidate note sequences are created using each possible parameter configuration. A single candidate is then selected, using a fitness function that compares the candidate note sequences with the target note sequence.

Recall that the number of transformations is m , that is $m = |C|$. Let j be an index into the list of transformations, C , ranging over $1 \leq j \leq m$. For C_j , there are a related set of parameter configurations, P_j . The parameter configurations within P_j are used by C_j to create transformed note sequences within a ‘selection pool’; let it be O_j . The 0^{th} item in each parameter set is “bypass”, $P_{j,0} = \text{bypass}$, thus enabling the unmodified input to also be an item in the pool.

For example, the *rate* transformation multiplies the start-time of each note by a ratio from the set $\frac{1}{4}, \frac{1}{2}, \frac{2}{3}, \frac{3}{2}, 2$ and 4 . If *rate* is the 1st transformation, that is, $C_1 = \text{rate}$, the corresponding parameter set is: $P_1 = \{1, \frac{1}{4}, \frac{1}{2}, \frac{2}{3}, \frac{3}{2}, 2, 4\}$. While *rate* has a single parameter, more complex transformations include particular combinations of multiple parameters. Having a fixed set of values may seem like an unnecessary limitation when dealing with parameters that are otherwise continuous; however, it also affords precise control over parameter values and thus musical style. As well as this, the parameter sets can be any size and requiring discrete specification encourages careful consideration of the musical ramifications of each parameter value.

Each transformation C_j has an input note sequence; let it be I_j . Let k be an index to parameters in the set P_j and the corresponding output in O_j , that is, $0 \leq k \leq |P_j|$. Thus, the creation of the k^{th} note sequence in the pool for the j^{th} transformation is: $O_{j,k} = C_j(I_j, P_{j,k})$. Continuing the *rate* example, where $j = 1$, creating the 1st note sequence in the selection pool ($k = 1$) would be: $O_{1,1} = C_1(I_1, P_{1,1}) = C_1(I_1, \frac{1}{4})$. In this example, the start time for each note would be multiplied by $\frac{1}{4}$ and the result looped four times to make it the same length as the input. The diagram below provides an example of the complete selection pool for this *rate* example:



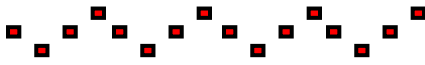





 <p>Example of an input to rate, I_1.</p>	 <p>Zeroth pattern in the selection pool, $O_{1,0}$ where $P_{1,0} = 1$ (bypass).</p>
 <p>First pattern in the selection pool, $O_{1,1}$ where $P_{1,1} = \frac{1}{4}$.</p>	 <p>Second pattern in the selection pool, $O_{1,2}$ where $P_{1,2} = \frac{1}{2}$.</p>
 <p>Third pattern in the selection pool, $O_{1,3}$ where $P_{1,3} = \frac{2}{3}$.</p>	 <p>The fourth pattern in the selection pool, $O_{1,4}$ where $P_{1,4} = \frac{3}{2}$.</p>
 <p>The fifth pattern in the selection pool, $O_{1,5}$ where $P_{1,5} = 2$.</p>	 <p>The sixth pattern in the selection pool, $O_{1,6}$ where $P_{1,6} = 4$.</p>

Figure 2 Each of the seven note sequences that would be in a selection pool for *rate* ($j=1$), given the original (top-left).

A fitness function determines which note sequence in the selection pool, O_j , is selected for output. Let the index of this selected note sequence in O_j be denoted by k^* . The input to the first transformation, C_1 , is the previous frame (in the first cycle, this will be the source). Recalling that n is the number of frames and that $1 \leq i \leq n$, this means: $I_1 = M_{i-1}$. For transformations other than the first in the chain, the input is the selected output of the previous transformation in the chain, that is, for $2 \leq j \leq m$, $I_j = O_{j-1,k^*}$. Each frame other than the first is derived from the selected output of the last compositional transformation in the chain, that is, for $2 \leq i \leq n$, $C_m(C_{m-1}(C_{m-2}(\dots(C_1(M_{i-1})))) = O_{m,k^*} = M_i$. The first frame is the source music: $M_1 = S$.

7.2.1 Selection through dissimilarity measures

Each element in the selection pool is compared with T using a dissimilarity measure¹. The selected note sequence, O_{j,k^*} , has a dissimilarity rating that is the closest to a particular target value. The process for determining the target value of dissimilarity with T is described further below. The application of user-defined weightings and “convergence” settings are also involved.

The dissimilarity measure for a particular transformation C_j , denoted by D_j , is designed to complement that transformation. For example, if $C_1 = \text{rate}$, D_1 calculates dissimilarities between the inter-onset envelopes, which are directly affected by the *rate* transformation. Each dissimilarity measure takes two note sequences and returns a value between 0 (similar) and 1 (dissimilar).

Each dissimilarity measurement between elements in the selection pool and the target is factored by a user controlled weight. The weights allow various parameter configurations to be unnaturally favoured which means the style of compositional transformations used throughout the morph can be influenced by the user.

¹ Discussing the measures in terms of ‘dissimilarity’ rather than similarity is an arbitrary choice, however, it is useful as ‘dissimilarity’ is analogous to ‘distance’, which is a natural method of comparing two values. The dissimilarity measure is not a distance metric in the formal mathematical sense, because triangle inequality is not necessarily upheld.

Let w hold the user-defined weights. Recall that k is an index to the items in the parameter set P_j and the corresponding selection pool O_j . Let each item in the selection pool $O_{j,k}$ have a related weight, $w_{j,k}$. Let $R_{j,k}$ hold the result of the dissimilarity between $O_{j,k}$ and T , scaled by $w_{j,k}$. This means: $R_{j,k} = w_{j,k} D_j(O_{j,k}, T)$. Each measure in the set of weighted results R_j can be sorted from lowest to highest. Let the result of this sorting be \vec{R}_j . Thus, within this list, $k = 0$ refers to the smallest value (least dissimilar), and $k = |R_j|$ refers to the largest (most dissimilar). The selection pool can be arranged in the same order, so that $\vec{O}_{j,0}$ refers to the $O_{j,k}$ that minimises $w_{j,k} D_j(O_{j,k}, T)$.

7.2.2 Speeding through the scenic route

If the goal of *TraSe* was to converge with T in as few frames as possible, then it would make sense to not use weights, and always choose $\vec{O}_{j,0}$, however, this is not the case. The aim instead is to find a *musical* progression and this involves taking the “scenic” route. The **transform speed** is a user-defined parameter, s_j , which influences the extent to which the number of frames, n , is minimised, for each of the transformations in the chain. s_j is between 0, for the maximised or “slowest” setting, and 1, for the minimised or “fastest”.

The inverse of s_j represents how many frames, n , can be expected using a single **perfect transformation**. A perfect transformation is a theoretical transformation that has the capacity to produce an infinite number of patterns in the selection pool, with an even spread of dissimilarity to T throughout. In a perfect transformation, when $s_j = \frac{1}{3}$, there will be 3 cycles before convergence. In practice, most transformations are imperfect and with some combinations of transformations and settings, convergence is never reached.

Target dissimilarity and transformation speed

From the user-defined transformation speed, s_j , a target value of dissimilarity, let it be t_j , needs to be established. This involves estimating how much the dissimilarity should be reduced by each step to ensure the specified number of steps, $1/s_j$, and multiplying it by the number of steps that have occurred already, i , to find the appropriate target, t_j , for this step. The estimate of the change in dissimilarity required for a single step is obtained by normalising s_j between the difference in least dissimilar, $\vec{R}_{j,0}$, which would be 0 with a **perfect** transformation, and the rating of the current note sequence that is the input, $R_{j,0}$ (recall that $P_{j,0} = \text{bypass}$). Considering that it would be unfeasible to pick a target that is below the lowest dissimilarity, $\vec{R}_{j,0}$ and recalling that

i is the current frame, t_j is defined thus:

$$t_j = \max \left(\vec{R}_{j,0}, R_{j,0} - i(R_{j,0} - (1 - s_j)(R_{j,0} - \vec{R}_{j,0})) \right)$$

Equation 1 Defining the target value of dissimilarity t_j . s_j is the user-defined ‘speed’ (0 is slowest). $\vec{R}_{j,0}$ is the lowest dissimilarity rating. $R_{j,0}$ is the dissimilarity rating of the unmodified input, that is, the previous frame.

This definition of t_j can be understood more intuitively thus: when the speed is set to slow, $s_j = 0$, it becomes $t_j = \max \left(\vec{R}_{j,0}, R_{j,0} - i\vec{R}_{j,0} \right)$. In the **perfect** case, where $\vec{R}_{j,0} = 0$ and $R_{j,0} = 1$, this becomes $t_j = \max(0, 1) = 1$. Note also that convergence does not occur in a single frame.

With this target (the most dissimilar), the algorithm will never converge, which is appropriate considering that as s_j approaches 0, $1/s_j$ (the number of steps) will approach ∞ . If the speed is set to fast, $s_j = 1$ we obtain $t_j = \max \left(\vec{R}_{j,0}, R_{j,0} - iR_{j,0} \right)$, which, in the **perfect** case, becomes $t_j = \max(0, 1 - i)$.

In this case, for $1 \leq i \leq \infty$, the target similarity will be 0. This means that, in the **perfect** case, the algorithm would converge in a single frame, confirming that $1/s_j = n$.

Aiming for target dissimilarity while maintaining consistency

An appropriate k^* (the index of the selected sequence) will minimise the difference of measured dissimilarity, R_{j,k^*} , to the target dissimilarity, t_j , while at the same time minimising the dissimilarity of the selected sequence O_{j,k^*} to the source, S . This ensures that in situations where the candidates are rated equally close to t_j , the one that is the most similar to S will be selected, providing a greater sense of musical continuity. This will involve interpolating between the consistency with source and target for the currently desired dissimilarity. The difference between $R_{j,k}$ and the target dissimilarity t_j is: $|t_j - R_{j,k}|$. The distance from the source is $D(S, O_{j,k})$. Let v_j be a user-defined weight, called ‘tracking VS consistency’, which controls the balance between tracking t_j , when $v_j = 0$, and being consistent with S , when $v_j = 1$. This effectively controls the influence of $|t_j - R_{j,k}|$ or $D(S, O_{j,k})$. Usually, $v_j < 0.5$ so the difference to the target dissimilarity is the principal component. All together, k^* is determined thus:

$$\min_{k^*} v_j |t_j - R_{j,k^*}| + (1 - v_j) D(S, O_{j,k^*})$$

Equation 2 The index to the selected note sequence, k^* , is determined by minimising the difference of the dissimilarity of that note sequence with the target dissimilarity (first term above), while minimising the dissimilarity between that note sequence and the source (second term). The user can control the influence of each of these terms using v_j - the ‘tracking VS consistency’ variable.

Capping the number of transformations per cycle

The ‘mutation limit’ is a cap that can be placed on the number of transformations applied in each iteration of the transform-chain and is controlled by the user. This should not be confused with the ‘**dissimilarity cutoff**’, which is essentially a threshold of fitness which stops the whole process, rather than just the transform-chain of a single cycle.

During each iteration of the chain, the number of transformations that result in a sequence other than the input, $O_{j,0}$, are counted. If the count exceeds the mutation limit, the chain finishes prematurely. Without the mutation limit, the extent to which the transform-chain reduces the dissimilarity of the input to T is unpredictable, as each subsequent transformation starts from the point at which the previous transformation left off. This effect can be reduced by the mutation limit. For example, a mutation limit of 1 will guarantee that only one transformation is used per cycle. Although this is currently sufficient, other improvements to the chain structure have been envisaged and are discussed below (7.7.3).

7.2.3 Putting it together

In pseudocode

A summary of the Transform-Select function of the *TraSe* algorithm is provided in pseudocode below (Figure 3):

```

// input:
C      a compositional transformation.
C.TRANSFORM takes a note sequence and a parameter setting and returns a transformed note
sequence.
C.P    is an array of different parameter settings that can be passed to C.TRANSFORM.
Note that one of the parameter settings is "bypass".
C.D    measures the dissimilarity of two note sequences, returning values between 0 (most
similar) and 1 (most dissimilar).
I      is the input note sequence that will be transformed by C
Src,Tar source and target note sequences(read-only).
i      the number of frames that have been generated so far
// user-defined global parameters:
W      an array of weights used to favour certain parameter settings in C.P over others
s      controls convergence speed by influencing the target level of dissimilarity.
v      "track VS consistency": controls the balance between minimising the difference of
the dissimilarity rating of the output to the target dissimilarity and minimising
the dissimilarity of the output to the source.
// output:
Out    a transformed sequence. If W is not used, C.D(Out, T) <= C.D(I, T)

function TRANSFORM-SELECT (transformation C, note sequence I, note sequence Src,
                           note sequence Tar, int i) returns Out {

    SP = new note sequence array, length of LENGTH(C.P) // the selection pool
    SP[0] = COPY(I)

    R = new double array, length of LENGTH(C.P) // the ratings
    R[0] = C.D(I,T)

    min = 0; // the index of the lowest rating

    FOR ( j = 1; j < LENGTH(P) ; j++ ) {

        SP[j] = C.TRANSFORM(COPY(I), C.P[j]) // add a transformed sequence
        R[j] = C.D(SP[j], T) // rate the transformed sequence
        R[j] = R[j]*W[j] // weight it

        IF(R[j] < R[min]) min = j // update the index to the smallest rating
    }

    // derive the target value of dissimilarity to T that is to be aimed for
    t = MAX(R[min], R[0] - i*(R[0] - (1 - s)(R[0] - R[min])))

    dmin // index of the pattern to be returned

    FOR( j = 0; j < LENGTH(P); j++ ) {

        // factor in the distance to t and dissimilarity with source
        R[j] = v*ABSOLUTE(R[j] - t) + (1 - v)D(SP[j], S)

        IF(R[j] < R[dmin]) dmin = j // update the reference to the minimum
    }

    Out = SP[dmin]
    return Out
}

```

Figure 3 Pseudo-code description of the Transform-Select function that is used in the *TraSe* algorithm.

Techniques for dealing with too many frames

Often there are too many frames in M , due to the transformations in C being far less able to converge than the theoretical **perfect** transformation. Imagine if the required morph is 32 beats

long and 64 frames were generated, or, $n = 64$. This means that, in the default approach, the entire morph will be created by 0.5 beat segments from each of the 64 frames, as $32 = 0.5 * 64$, which is unlikely to exhibit much coherence.

A simple method to counter this is to quantise the morph index, Ω , so that only a particular number of frames will be played. For example, if the user specifies 4 sections, Ω will only ever be one of the four numbers in the list: $0 + \frac{1}{8}, \frac{1}{4} + \frac{1}{8}, \frac{1}{2} + \frac{1}{8}, \frac{3}{4} + \frac{1}{8}$. The $\frac{1}{8}$ is added so that Ω will be central to each $\frac{1}{4}$ section, not at the beginning of it. In this example, the only frames that will be played are $64 * \frac{1}{8} = 8$, $64 * \frac{3}{8} = 24$, $64 * \frac{5}{8} = 40$ and $64 * \frac{7}{8} = 56$. From each of these frames, 8 beats will be played, which allows more coherence than the 0.5 beats that would occur without quantization.

However, with so many cycles separating each frame (16 in the example given), the sense of a logical progression may be lost. This can be minimised somewhat with carefully designed compositional transformations. For example, the *add/remove* transformation (see 7.3.8) is a special transformation that is always included at the end of the compositional transformation chain in order to ensure this. *Add/remove* is designed in such a way as to guarantee an output, M_i , that is closer to the target, T , than the input, I_n . In fact, this transformation is necessary due to the potential lack of convergence of the algorithm.

7.2.4 Summary of transforming and selecting

In summary, *TraSe* involves passing the source through a series of transformation and selection cycles until it becomes identical, or almost identical, to the target. The architecture provides the ability to favour certain parameter configurations which in turn allows greater control over musical style. The limitations inherent in *TraSe* are closely aligned with morphing:

- **Smooth** transitions are afforded through the fact that any one frame is more closely related to its neighbours than any other, in terms of the number of applications of the transformation chain.
- Transitions that are **coherent** for a particular style of music are afforded through the ability to include and favour transformations and parameter configurations that are indicative of that style.
- Musical material beyond source and target is not required.

During the above explanation of transforming and selecting, I have alluded to various

compositional transformations such as *rate* and *add/remove*. The following section will detail each of the different compositional transformations that have been developed and are implemented in *TraSe*.

7.3 Specific compositional transformations: their process, parameters and dissimilarity measures

The transformations that have been implemented are described here in order of their position in the transformation chain: *divide/merge*, *rate*, *phase*, *harmonise*, *scale pitch*, *inversion*, *octave*, *add/remove* and *key/scale*. All of these perform large scale transformations to the note sequences, except for *add/remove*, which deals with individual notes. *Key/scale* morphing is a special transformation which operates in parallel to the other transformation, using key and scale data rather than note sequence data. This set of compositional transformations were chosen due to their capacity to affect musical dimensions such as note density, rhythm, harmony, melody, register, and tonality. There are no doubt other compositional transformations that may improve the performance of the algorithm, however, the current set is sufficient.

Each transformation has a range of parameter configurations which are used to generate a pool of potential note sequences for that transformation, each time it is called. The candidate note sequences are rated according to their dissimilarity with the target, using a dissimilarity measure which is specific to that transformation.

7.3.1 Divide/merge

The *divide/merge* transformation was envisaged as a technique for affecting the note density of the input without dramatically altering the music. *Divide/merge* has five parameter configurations, apart from 'bypass', and uses a 'Nearest Neighbour' dissimilarity measure to compare the transformed items in the selection pool with T . The five configurations are: merge forwards, merge backwards, split $\frac{1}{4}$, split $\frac{1}{2}$ and split $\frac{3}{4}$.

'Merge forwards' iterates through the note sequence, merging notes that overlap into one, retaining the pitch and dynamic of the earlier note. The end-time of the earlier note will increase to become the end-time of the later note and the later note will be removed. 'Merge backward' is similar, except that it proceeds from the last note to the first and the scale degree and dynamic are copied from the later note, deleting the earlier note.

```

// inputs:
I      the input note sequence
// output:
M      the note sequence that is the result of the merging process on I
MERGE-FORWARDS(note sequence I) returns M {
    M = COPY(I)
    Note curr, next          // a reference to the current and next notes

    FOR(int i = 1; I < LENGTH(M); i++) {
        curr = M[i-1] // update the current note
        next = M[i]   // update the next note

        // see if the duration of the current note exceeds the inter-onset
        IF(curr.onset + curr.duration >= next.onset) {

            // set the duration of the foremost note
            SET-DURATION(curr, next.onset - curr.onset + next.duration);

            REMOVE(M, next) // remove the note that was merged
        }
    }
}

```

Figure 4 Pseudocode of the Merge-Forward algorithm

‘Split’ finds the longest note and splits it in two. The three different parameter settings, $\frac{1}{4}$, $\frac{1}{2}$ and $\frac{3}{4}$ refer to the point at which the note is split. For example, splitting a whole note (four beats) with $\frac{1}{4}$ would turn it into one crotchet followed by a dotted minim. The scale degree and dynamic for both of these notes would be the same as the original.

To compare each note sequence generated by the difference parameter settings, *Divide/merge* uses the NN dissimilarity measure (described below). ‘Average number of notes’ was trialed originally, as *divide/merge* was aimed primarily at controlling the number of notes. However, this was not discerning enough to enable convergence with the target and so NN has been used satisfactorily instead.

A more flexible and natural design for *divide/merge* might be achieved through an envelope representation, as described in the parametric morphing algorithm chapter. With the input as envelopes, merging would be akin to smoothing, while dividing would be similar to interpolating between the points.

The Nearest Neighbour (NN) dissimilarity measure

The Nearest Neighbour (NN) dissimilarity measure is the most thorough of all dissimilarity measures within this research. For each note in the two note sequences being compared, the NN measure finds the distance of the closest (neighbouring) note in the opposite sequence. The final distance returned is the average of all such distances. This is an inefficient process and some ideas for optimisation are explained below (7.7.6).

The process for NN is bidirectional, comparing the target sequence of notes, T , to the input, I , as well as I to T . These two calculations will be referred to as **backward** and **forward** respectively. The bi-directionality is necessary because the NN of one is not necessarily the NN of the other. This is made clear in Figure 5 where the first note in I is closest to the first note in T but the first note in T itself is closer to the second note in the input than the first:

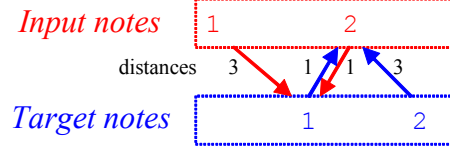


Figure 5 A bi-directional NN comparison. The red box holds two notes (red 1 and 2) from the input sequence. The blue box holds two notes (blue 1 and 2) from the target sequence. The blue arrows show the NNs from the *backward* computation and the red arrows show the NNs from the *forward* computation. The distances between NNs are adjacent to the arrows.

Let A and B be two note sequences and let $av(A, B)$ be the average distance between each note in A and its NN B . Let i be an index to A that ranges over $1 < i \leq |A|$ ($| \cdot |$ indicates length) and let j be an index to B that ranges over $1 < j \leq |B|$. Let $dist$ be a function that finds the distance between two different notes primarily in terms of pitch and note onset (described above in 7.3). The av function is thus:

$$av(A, B) = \frac{1}{|A|} \sum_{i=1}^{|A|} \min_{j=1}^{|B|} (dist(A_i, B_j))$$

Equation 3 The average distance between each note in A and its NNs in B .

The *forward* calculation will be $av(I, T)$ and the *backward* calculation will be $av(T, I)$. Letting nnd be the NN dissimilarity measure, we have:

$$nnd(I, T) = \frac{1}{2} (av(I, T) + av(T, I))$$

Equation 4 The NN dissimilarity measure.

The note distance function for the NN measure is a combination of distance between note onsets within the loop and distance between “octavised” scale degrees. That is, the *DePa* scale degree, d , plus the octave for that pitch, o , multiplied by the number of *DePa* scale degree

steps per octave, $s: d + os$. This scheme favoured the use of source and target note sequences within the same octave. In future work it would be simple to overcome this by calculating the octave and scale degree distance separately and combining them afterward, weighted on a user-defined variable. Other measures that compare the similarity of scale degrees and passing notes could be applied. Inclusion of duration and dynamic and the Circle of Fifths (CF) distance, as explained in the previous chapter, would also be simple.

Pseudocode of the NN dissimilarity measure is included below:

```
// input:
M      A note sequence being used for comparison (often the result of a transformation)
T      A second note sequence being used for comparison (often the Target)
// global functions:
FIND-DISTANCE
    takes two notes and returns the distance between them in octavised scale degree
    and note onset.
// output:
avd    The average distance to nearest neighbours
NND(M, T) returns avd {
    avdM, avdT, avd// stores average NN distance for M, T and the final average

    FOR (j=0; j < LENGTH(M); j++) { // find the neighbour of each note in M, in T

        min_d // reference to minimum distance

        FOR (i=0; i < LENGTH(T); i++) {
            distance = FIND-DISTANCE(T[i], M[j])
            IF(distance < min_d) min_d = distance
        }
        avdM += min_d // accumulate the average
    }

    avdM = avdM/LENGTH(M) // normalise

    FOR (i=0; i < LENGTH(T); i++) { // find the neighbour of each note in T, in M

        min_d // reference to minimum distance

        FOR (j=0; j < LENGTH(M); j++) {
            distance = FIND-DISTANCE(T[i], M[j])
            IF(distance < min_d) min_d = distance
        }
        avdT += min_d // accumulate the average
    }

    avdT = avdT/LENGTH(T) // normalise

    avd = (avdM + avdT)/2 // final average
}
```

Figure 6 Pseudocode of Nearest Neighbour dissimilarity measure

7.3.2 Rate

Rate multiplies the onset of each note by a ratio, removing notes that exceed the loop length, or looping the sequence as many times as needed to fill the loop. The ratios are based on patterns from MEM (Mainstream Electronic Music) genres: $\frac{1}{4}$, $\frac{1}{2}$, $\frac{2}{3}$, $\frac{3}{2}$, 2 and 4. For example, a straight

beat transformed by $\frac{3}{2}$ or $\frac{2}{3}$ yields a common three against four rhythm. Transformations by $\frac{1}{2}$, and $\frac{1}{4}$ are notorious in breakdowns, while a rate change of 2 or 4 often increases the intensity of build-ups.

The dissimilarity measurement used for *rate* is the difference in area between the onset envelopes, combined with the difference in area between the pitch envelopes (for an example, see Figure 7, below). The difference for inter-onset is normalised by the maximum possible difference area, which is the length of the loop squared. The difference for pitch is normalised by an arbitrary maximum of 30 multiplied by the loop length, which is equivalent to one note in each pattern, 30 semi-tones apart. If the difference happens to be greater than this, the maximum dissimilarity of 1 is returned. These measures are sufficient, but could be improved by normalising by the maximum (for pitch only) and magnifying the lower end of the spectrum through a logarithmic function, as implemented in the “linear pitch similarity” measure of the *Markov Morph*.

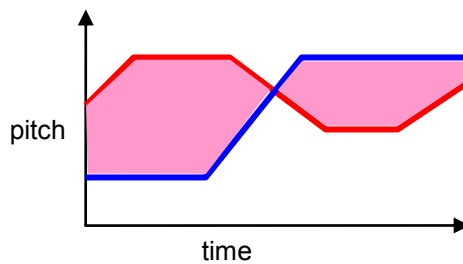


Figure 7 Two pitch envelopes (red and blue) and the difference in area between them (in pink). The difference in area between the onset and pitch envelopes are used as the dissimilarity measure for *rate*.

The result is probably similar to a NN measure. A comparison of the two measures is yet to be thoroughly investigated, however it is imagined that because *rate* preserves the contour of the envelope, the dissimilarity function that measures envelope differences is more relevant than the NN distances.

7.3.3 Phase

The *phase* transformation shifts the start time of each note within the input by a certain amount, either ahead or behind. Onsets that exceed the loop boundary are wrapped around. The dissimilarity measure for *phase* is the NN dissimilarity measure, because of the high-level of accuracy in absolute difference between notes. The effect of *phase* is that the input pattern is “nudged” ahead or behind a certain amount, so as to maximise the absolute similarities between

the patterns. This can have the musical effect of highlighting the similarities of notes relative to one and other (rather than relative to the metre) in the input and target patterns. The parameter space for *phase* is the series of quarter-intervals from -4 through to $+4$, that is, $-4, -3.75, -3.5, \dots, 3.5, 3.75, 4$. This makes 32 different parameter settings, not including 0 or “bypass”.

7.3.4 Harmonise

Harmonise works by either adding or removing parallel harmony at each of the tonal intervals from the 3^{rd} through to the octave. This means 12 different parameter settings in all, excluding “bypass”: remove $8^{ve}, 7^{th}, 6^{th}, 5^{th}, 4^{th}, 3^{rd}$; or add $8^{ve}, 7^{th}, 6^{th}, 5^{th}, 4^{th}$ or 3^{rd} . While not all of these intervals are particularly common, they are included to provide flexibility.

To remove a harmonic interval, the note data is analysed to find the notes that occur on the same start time but with a different pitch – “harmonic clumps”. Within each clump, notes that are the specified interval above the lowest pitch are removed. For the addition of harmonic intervals, monophonic notes are doubled with a parallel harmony at the specified interval.

The dissimilarity measure used by *harmonise* is a weighted combination of the difference in average harmonic interval and the difference in average clump size. The average harmonic interval is the sum of the average harmonic interval of each clump, normalized by the number of harmonic clumps (including clumps with a single note).

The weighting between average harmonic interval and average clump size is 0.8 and 0.2 respectively. This is because the most important effects rendered by *harmonise* are the changes in the harmonic interval, but clumps of various sizes can have the same average interval and so the average clump size is included so as to help differentiate between cases such as this.

7.3.5 Scale pitch

The *scale pitch* transformation expands or reduces pitch range, while maintaining contour. The pitch of each note is scaled by a certain ratio, relative to a Central Tonic (CT) pitch. There are fourteen ratio values, excluding “bypass”, in the set of possible parameter configurations. These are all the intervals of $\frac{1}{7}$ from 0 to 2: $\frac{0}{7}, \frac{1}{7}, \dots, \frac{13}{7}, \frac{14}{7}$. This was chosen so that it would be possible for an octave to be shifted to any of the other scale degrees above or below it.

The CT pitch is the tonic in the most central octave covered by the pitches of the input sequence. Firstly, the average pitch of all the notes is calculated and rounded down to the

nearest tonic. The pseudocode to find the CT pitch is as follows:

```
// Inputs:
I      an array of pitches
spo    the number of steps per octave.
// Output:
ctp    the tonic pitch of the most central octave
FIND-CENTRAL-TONIC(integer array I, integer spo) returns integer ctp {
    avp = SUM(I)/LENGTH(I)
    ctp = avp - MODULUS(avp, spo)
    return ctp
}
```

Figure 8 **Pseudocode for the algorithm to find the tonic of the central octave.**

For each note pitch in the pattern, the interval with the CT is found by subtracting the CT from it. This interval is multiplied by the scaling ratio and the fundamental is added back to it. The scaled pitch is then adjusted to be in-scale or out-of-scale, to be consistent with the original *DePa* value.

Finally, a bounding function sets the pitch to be the maximum if it is above maximum, or sets it to be the minimum if it is below minimum. The default bounds used in the *DePa* representation is 10 octaves: between 0 and the number of *DePa* steps per octave multiplied by 10. This would be 140 for a standard church mode. Pseudocode of *scale pitch* follows:

```

// Inputs:
I      an array of pitches (in DePa)
r      a ratio which controls to what extent the pitches are scaled
spo    the number of steps (in DePa) per octave.
hs     the largest number of consecutive semitone steps between each tonal semitone in
        the current scale
// Global functions:
BOUND  takes a DePa pitch and spo and constrains the pitch within the range 0 and spo*10
// Outputs:
I      the original array of pitches that has been scaled
SCALE-PITCH(integer array I, double r, integer spo, integer hs) {

    ct = FIND-CENTRAL-TONIC(I); // (defined above)
    FOR(int i = 0; i < LENGTH(I); i++) {

        // record the scale remainder. 0 is in scale, others are out of scale

        ton = MODULUS(I[i], hs)
        I[i] = I[i] - ct           // find difference with the fundamental

        I[i] = I[i] * r           // scale the difference

        // find the scale remainder of the scaled pitch
        nton = MODULUS(I[i], hs)

        // shift to the nearest point with a similar remainder
        IF( ABSOLUTE(ton - nton) < ABSOLUTE(ton + hs - nton) ) {
            I[i] += ton - nton
        } else {
            I[i] += ton + hs - nton
        }
        I[i] = I[i] + ct           // add the fundamental back in
        BOUND(I[i], spo)           // constrain it within bounds
        return I[i]
    }
}

```

Figure 9 Pseudocode for the *scale pitch* compositional transformation.

An alternative *scale pitch* technique that avoids the need to bound the pitches is to determine the maximum possible ratio without exceeding the bounds, and weight this ratio with a parameter to determine the final scaling ratio that is used. This approach was trialed, however, the current approach was favoured as it was more predictable and the exceeding of bounds was rarely a problem.

The dissimilarity measure used for *scale pitch* is the difference in average interval from the CT, $avc(I)$. Recalling that I is a sequence of note pitches, n is the length of I , i is an index to I with the range $0 \leq i < n$ and c is the central tonic pitch, we have:

$$avc(I) = \frac{1}{n} \sum_i |I_i - c|$$

Equation 5 The average interval from the central tonic. I is the sequence of note pitches, n is the number of note pitches in I (the cardinal). i is an index to I and c is the central tonic.

This measure is used because the average interval from the central tonic is directly affected by the *scale pitch* function. One improvement to this measure could be to store and use the CT of the original sequence, rather than recomputing it for each of the scaled sequences. This is because in some situations the CT itself may be shifted, resulting in overly dramatic scaling ratios, as evident below (7.4.1).

7.3.6 Inversion

Inversion operates in a similar way to “chord inversion”, but inverts the pitches of the input sequence, I , rather than a chord. The degree of inversion is controlled by a parameter, let it be p , that ranges between $-1 \leq p \leq 1$. The number of octaves, q , by which to shift the pitches that are to be inverted must be greater than the octave range of the sequence, so that inverted notes do not clash with existing notes. Letting the number of steps per octave be denoted by s , we have $q = \frac{1}{s} (\max(I) - \min(I)) + 1$, rounded down².

The parameter p controls which pitches will be shifted and the direction. $|p|$ is the fraction of the range that will be shifted, and $\text{sign}(p)$ determines whether the percentage is taken from the top or bottom of the range and which direction it will shift. When $p > 0$ the selected pitches are taken from the bottom and shifted up and when $p < 0$ they are taken from the top and shifted down. For example, when $p = \frac{1}{2}$, all of the pitches that are below half the pitch range of the input pattern are shifted up by q octaves and when $p = -\frac{1}{2}$, all of the pitches that are above half the pitch range are shifted down by q octaves. The *inversion* transformation is in pseudocode below:

² Adding 1 and rounding down is chosen over simply rounding up, as it includes the octaves, which would not normally be rounded up.

```

// Inputs:
I      an array of note pitches that are to be inverted
p      fraction of the range of pitches and the direction of the inversion
spo    the number of steps in each octave
// Outputs:
I      the array of note pitches that has been inverted

INVERT(integer array I, double p, integer spo) { returns integer array I
    lop = FIND-LOWEST(I)
    hip = FIND-HIGHEST(I)

    shift = (ROUND-DOWN((hip-lop)/spo) + 1) * spo           // the amount to shift

    FOR(integer i = 0; i < LENGTH(i); i++) {
        IF( p > 0) { // inverting upwards
            IF( I[i] <= ((hip-lop) * p + lop) ) I[i] = I[i] + shift
        } ELSE-IF(p < 0) { // inverting downwards
            IF( I[i] >= (hip-lop) * (p + 1) + lop ) I[i] = I[i] - shift
        }
    }
    return I
}

```

Figure 10 Pseudocode of the inversion transformation.

To select an inversion, the dissimilarity measure used is the difference in pitch envelopes, as described above in *Rate* (7.3.2). This was chosen because the inversion process has a substantial effect on the contour of the pitch envelope.

7.3.7 Octave

The *octave* transformation shifts the whole note sequence by a number of octaves, either: -3 , -2 , -1 , 1 , 2 or 3 . The dissimilarity measure used for octave is simply the difference in average pitch.

7.3.8 Add/remove

The *add/remove* transformation guarantees that the output will be closer to the target, T , than the input, I . Each note in I is considered for removal and each note in T is considered for addition³ and because of this, the parameter space for *add/remove* is not fixed like the other transformations. The NN dissimilarity measure is used to compare each result to T , so as to ensure accurate results. *Add/remove* is always the last transformation in the chain, enabling

³ Adding notes from the target is a kind of ‘cheat’. Ultimately, *TraSe* aims to function without deriving material directly from the target.

sequences that are not brought closer to \mathcal{T} by any other transformation to eventually converge. Add/remove can be cycled a number of times specified by the user, feeding the output immediately back into the input. This is the 'add/remove cycles' parameter.

Add/remove operates in either 'polyphonic' or 'monophonic' mode. In monophonic mode, if a note is added at an onset that is already occupied, the existing note is replaced by the new note. Specifically, the pitch of the old note is replaced by that of the new note whereas the values of duration and dynamic for the new note are derived through combination between the old and new notes. The weighting of the combination is determined by the user. This feature was added after I had trialled source and target examples with very different durations and dynamics. It was found to add a degree of smoothness to the transition.

In polyphonic mode, the new note is overlaid, regardless of whether a note exists on the same onset. If a note exists on the same pitch (and onset), the attributes of the existing note are merged with the new note, using the same weighted combination parameter as monophonic mode.

Monophonic mode has greater transformational impact than polyphonic mode because, through replacement, an addition and removal can occur in a single step. Musically, especially in contexts that are mostly monophonic, the harmonies generated by polyphonic *add/remove* can sound unrealistic. A possible improvement could be for polyphonic mode to perform complete vertical replacements. That is, add or remove vertical note groups rather than single notes.

7.3.9 Key/scale morph

The *key/scale* morph deals with key and mode data, rather than sequences of notes, and operates in parallel to the note sequence transformations described above. In other words, each *TraSe* morph consists of a sequence of note pattern frames and a separate sequence of *key/scale* frames, which can be of a different length.

The frames generated by *key/scale* morph each contain a key which can be from C, C#, and so on, through to B, and a scale which can either be ionian, dorian, phrygian, lydian, mixolydian, aeolian, locrian or harmonic minor. The selection pool at each step of the key/scale search consists of the 96 different combinations of 12 keys and 8 scales.

If the parameter controlling the speed of transformation is set to 1, the process will converge in a single step because the parameter space covers the complete range of values that are possible. This is close to being the **perfect** transformation mentioned in 7.2.2. The transform speed is

typically set at less than 1, so that a progression of key and scale changes is generated.

The dissimilarity measure used for comparing of each of the 96 key-scales with the target is a weighted combination of *scale dissimilarity*, *key-scale dissimilarity* and *key-root-distance*. The user specifies the weights manually. *Scale dissimilarity* takes two different scales, each consisting of a set of allowable pitch classes, and counts how many are not shared, normalizing the result between 0 and 1. This is a similar technique as used by Polansky (1987). *Key-scale dissimilarity* is similar to *scale dissimilarity* except that the pitch-classes in the scales are first transposed by the key. For example, with *scale dissimilarity*, C-Ionian and A-Aeolian would be considered different by a factor of $\frac{3}{7}$ (the 3 comes from the difference between Ionian and Aeolian in the 3rd, 6th and 7th), whereas with *key-scale dissimilarity* there would be no difference at all, because A-Aeolian is the relative minor of C-Ionian and the two pitch class sets are identical.

Key-root distance is a weighted combination of distance between keys in the CF and the Circle of Chroma (CC), both of which were explained in detail in the previous chapter. Let s represent the number of steps per octave, and thus the number of keys also. The default implementation is $s = 12$, with the keys roots from 0 (C) to 11 (B), however, it is possible to use other tuning systems and scales. If a and b are the two keys roots being compared, then the distance between them in CC space is an application of the circle distance function, $ccd(a, b, s) = \frac{2}{s} \min((a - b) \bmod s, (b - a) \bmod s)$. The CF distance is similar, except that a and b are first transformed into the CF by multiplying by a fifth, modulo s .

In all, there are five different weights that influence dissimilarity measurement: scale dissimilarity, key-scale dissimilarity, key-root distance, CC key-root distance and CF key-root distance. Tweaking these weights, along with the morph speed, is the primary method for the user to control the musicality of *key/scale morph*. These parameters enable a surprising degree of musical control. For example, a “pop” style key change which retains the scale and shifts by chromatic increments can be achieved by favourably weighting scale dissimilarity, key-root distance and CC key-root distance. A more jazz-oriented key modulation, that walks through the CF and utilises pivot notes can be achieved by favourably weighting key-scale dissimilarity, key-root distance and CF key-root distance.

7.4 Informal evaluation: listening and analysis

Informal evaluation of *TraSe* was an iterative process of morph generation, reflection, parameter adjustment and further morph generation. From this, pertinent examples have been selected and

are discussed below. Firstly, a series of morphs that are produced from the same source and target note sequences but with different parameter tunings are analysed and compared, providing some insight into the *TraSe* process and the effect of the parameters. Following this, some differences between morphing forwards and backwards are examined. Some examples of *key/scale morphing* are also shown, including ii-V-I style modulation as well as direct chromatic shifts. The section concludes with a morph tribute to Mathews and Rosler (1969), which, due to the length of the note sequences is a particularly difficult example.

7.4.1 Tuning TraSe parameters

By tuning the parameters of *TraSe*, a range of different musical outcomes are possible. The first example below does not bias any particular transformation, however, it appears to have split too many notes with the *divide/merge* transformation, resulting in a degree of chaos. The second example biases note merging in order to balance note division. The third example demonstrates the emergent nature of *TraSe*: the transform speed is reduced, however, more frames are generated unexpectedly. The final example shows how smoother morphs can be obtained by limiting the number of transformations per cycle.

The source and target sequences that were used for all morphing examples have the same key and scale labels of C Mixolydian, and one bar in length. Although this automatically increases the similarity of the two, it was done so that the focus is on note sequence morphing rather than *key/scale* morphing or other aspects.

The source ([~7.1](#)) is a playful tune with short staccato notes played on 1.5 beat intervals which, against the 4/4 drums, creates a “three over four” type polyrhythm with a short macrocycle. It contains pitches C, D, F and B^b and the tonal movement is I-IV. In contrast, the target ([~7.2](#)) is an ambiguous, rising tonal wash with long notes that are mostly syncopated, a greater pitch range and a mostly upward contour. Two pitches are different to the source, E and G, while F is not included. The tonal movement could be interpreted as ii-I, VII-I, V-I or possibly other combinations. They both have six notes.

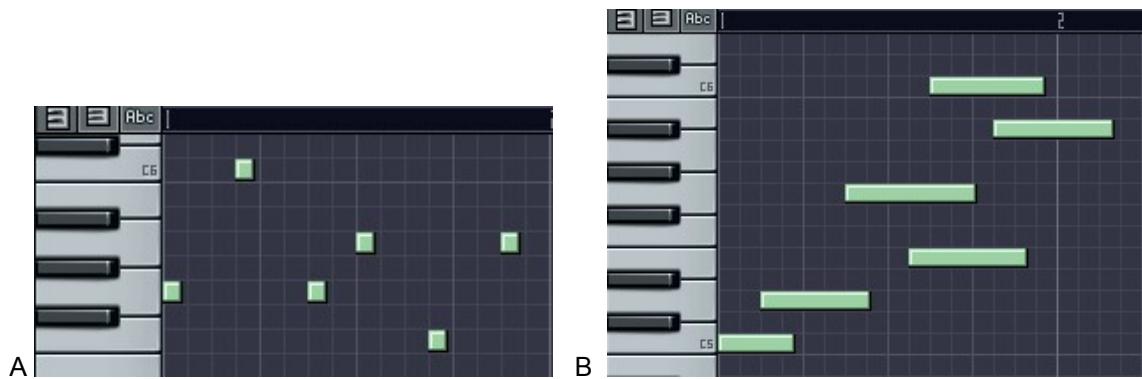


Figure 11 Close up of Source (A) and Target (B) .

Excessive note division inducing chaos

For this example ([~7.3](#)), all of the parameter weights were neutral, however, a slight excess of note division from the *divide/merge* transformation appears to have occurred. In order to obtain convergence at 6 frames⁴, the cut-off similarity rating for convergence was increased to 0.04 and the transform speed of *add/remove* was adjusted to 0.26. *Add/remove* was set to 2 cycles.

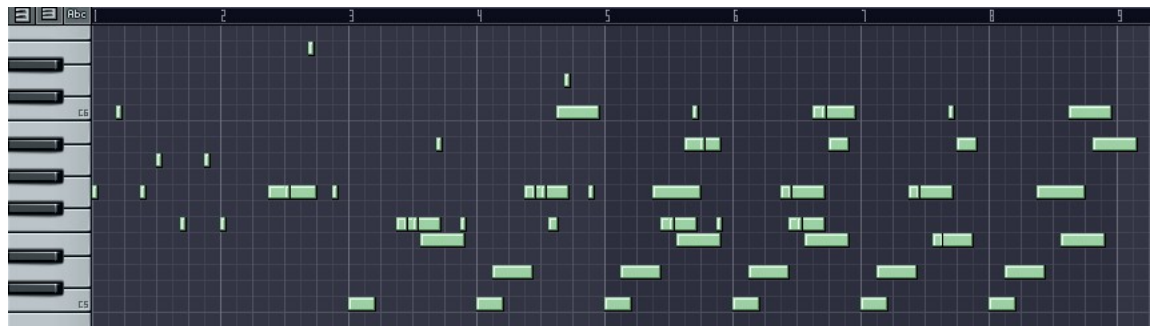


Figure 12 *TraSe* morph with all weights equal. *Add/remove* has two cycles, the transform speed of *add/remove* is 0.26 and the threshold of similarity to target is 0.04.

In the **first** frame, *scale pitch* shifted the G5s on 1.0 and 2.5 to F5, the C6 on 1.75 to A5, the A5s at 3.0 and 4.5 to G5 and the F5 at 3.75 to E5. Following this, the *inversion* transformation was applied upwards to a small degree, which had the effect of shifting E5 (the lowest pitch) at 3.75 up an octave to E6. The first cycle of *add/remove* replaced the F5 at 2.5 with the G5 of the target which also has a longer duration. The second cycle of *add/remove* removed the A5 on 1.75.

⁴ More than 6 frames becomes difficult to present.

In the **second** frame, *divide/merge* split the G5 on 2.5 into two notes, the second one appearing at 2.875. Pitch stretch shrunk the pattern, shifting all of the notes down by one tonal step, except for the E6 at 3.75 which was shifted down by three, to B^b. The first cycle of *add/remove* added the C5 at 1.0, while the second added the E5 at 3.25.

In the **third** frame, the E5 at 3.25 was split by *divide/merge*, resulting in another E5 at 3.5. A slight *scale pitch* was applied, shifting most of the notes up one tonal step and the B^b5 at 3.75 up two tonal steps to D6. The C5 at 1.0, remained in place. *Add/remove* added the D5 at 1.5 and replaced the new F5 at 3.5 with C6.

In the **fourth** frame, *divide/merge* split the C6, creating a new C6 at 4.125. *Scale pitch* applied a slight shrink, shifting the G5s at 2.5, 2.875, 3.0 and 4.5 down to F5, the C6s at 3.5 and 4.125 down to B^b5 and the D6 at 3.75 to C6. The first cycle of *add/remove* replaced the F5 at 3.25 with E5 and the second cycle of *add/remove* replaced the F5 at 2.5 with a G5.

In the **fifth** frame, the G5 at 2.5 was split by *divide/merge*, resulting in a new G5 at 2.185. The first cycle of *add/remove* replaced the B^b5 at 3.5 with C6, while the second cycle removed the F5 at 4.5.

In the **sixth** and final frame, the E5 at 3.25 was divided, resulting in another E5 at 3.5. The *harmonise* transformation removed the C6 at 3.5. The first cycle of *add/remove* removed the F5 at 2.875 and the second cycle removed the F5 at 3.0. If the similarity cut-off for convergence had not been increased to 0.04, the process would have continued for an additional 5 cycles.

The shrink applied by *scale pitch* in the first frame, in combination with the removal of the A5 at 1.75, seems to change the harmonic movement from I-IV to IV-V. The first frame also provides more potential for change due to the lower note density. In the second frame the harmony reverts to I-IV, with some additional highlighting of the third and seventh intervals brought about by note addition and pitch-shrinking respectively. In the third frame, the tonal centre becomes the ii and a degree of 'openness' results from the combination of whole-tones, perfect fourths and the greater pitch range. Rhythmically, the pace of frame three is more frantic, as the density increases by two from the original six, to eight. This increases further in frames four and five, which have ten notes, while the harmony becomes more ambiguous. In frame six, the melodic contour is essentially the same as the target, but with repeated notes on the G5 and E5. Overall, this morph appears to be somewhat chaotic due to the excessive note density created through note division.

Balancing note division with note merging

This example ([~7.4](#)) demonstrates how the excess note division of the previous example can be counter-balanced by adjusting weights to bias merging over splitting. In this case, all ‘splitting’ weights for *divide/merge* were set to 0.8, while ‘merging’ weights remained at 1.0. The transform speed for *add/remove* was 0.46 and the dissimilarity cut-off was 0.03 to ensure convergence in 6 frames.

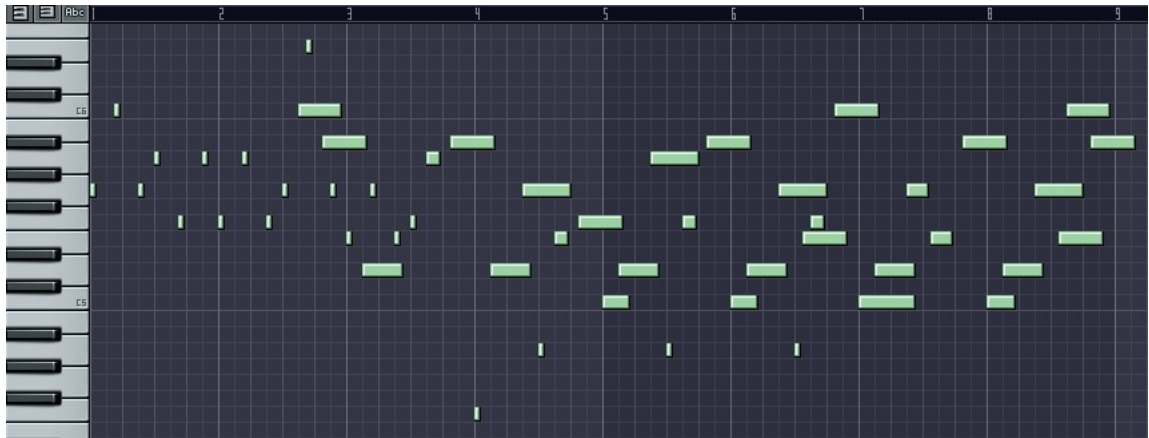


Figure 13 *TraSe* morph that demonstrates the use of weights that favourably bias the compositional transformation of “merging” notes over that of “splitting” notes, thus creating a less chaotic morph compared to the previous example. *Add/remove* transform speed is 0.46 and dissimilarity cut-off is 0.03.

In the **first** frame, *scale pitch* shifted all of the notes down one tonal step, except for the C6 at 1.75 which was shifted down by two. *Inversion* was then applied, shifting the E5 at 2.75 up an octave to E6. These two transformations are identical to those of the previous example. Unlike the previous example, *add/remove* added two notes, the C6 at 3.5 and the B^b5 at 4.25.

In the **second** frame, *divide/merge* applied a forward merge, fusing the E6 at 3.75 into the C6 at 3.5 and the G5 at 4.5 into the B^b5 at 4.25. Following the merge, *scale pitch* shifted all of the notes down one tonal step, except for the C6 on 3.5 and the B^b5 on 4.25, which were shifted down two tonal steps to A5 and G5 respectively. The first cycle of *add/remove* added the D5 at 1.5 and the second cycle replaced the G5 on 4.25 with B^b5.

In the **third** frame, *divide/merge* merged forward again, consolidating the G5 at 1.75 into the D5 at 1.5. This was followed by a severe *scale pitch* that shifted the E5 at 1.0 to F5, the D5 at 1.5 to E5, the E5 at 2.5 to F5, the F5 at 3.0 to A5, the A5 at 3.5 to E6 and the B^b5 at 4.25 to F6. This

severe stretch occurred because it shifted the central tonic to be C6 rather than C5. Recall that the similarity measure used by *scale pitch* is based on the average distance of pitches from the fundamental pitch. The average distance from C6, post stretch, was calculated as being closer than that of the previous average distance from C5. Following this, the whole pattern was then shifted down by an octave. *Add/remove* then replaced the F4 at 2.5 with G5 and the E4 at 1.5 with D5.

To generate the **fourth** frame, a slight downward *scale pitch* was applied, shifting the F4 at 1.0 down to E4, the G5 at 2.5 to A5, the E5 at 3.5 to F5 and the F5 at 4.25 to G5. The first cycle of *add/remove* replaced the E4 at 1.0 with C5 while the second cycle replaced the F5 at 4.25 with B^b5.

In the **fifth** frame, *scale pitch* shrunk the pitches slightly, resulting in the A5 at 2.5 shifting to B^b5 and the B^b5 at 4.25 shifting to C6. The first cycle of *add/remove* replaced the newly shifted B^b5 (at 2.5) with G5, while the second cycle added an E5 at 3.25.

In the **sixth** and final frame, a *divide/merge* applied a merge-forward, consolidating the D5 at 1.5 into the C5 at 1.0, the A4 at 3.0 into the G5 at 2.5 and the F5 at 3.5 into the E5 at 3.25. The first cycle of *add/remove* added the D5 at 1.5, while the second cycle replaced the C6 on 4.25 with B^b5.

The favourable weighting of merging over splitting in this example, and possibly the higher transform speed of *add/remove*, sets a different morphological trajectory to that of the previous example. The first frame is similar to that of the previous example, with an identical *scale pitch* and *inversion*. However, from the second frame onwards, there are numerous examples of merging where the previous example would divide, resulting in a less chaotic morph overall. The drastic *scale-pitch* of the third frame could be solved fairly trivially, as explained in 7.3.5, by referring to the original CT (Central Tonic) pitch of the source, rather than recalculating it for each transformed example. Alternatively, an additional transformation could be added after *scale/pitch* to compensate for the change of CT pitch (while maintaining the average distance from CT that was introduced by *scale/pitch*).

Emergent phenomena

This example ([~7.5](#)) demonstrates a situation where convergence unexpectedly occurred in fewer frames after the transform speed was reduced. In the previous example, a transform speed of 0.46 with dissimilarity cut-off of 0 yielded convergence in 9 frames, whereas, in this

example, when the transform speed was reduced to 0.28 from 0.46, it converged in 4:

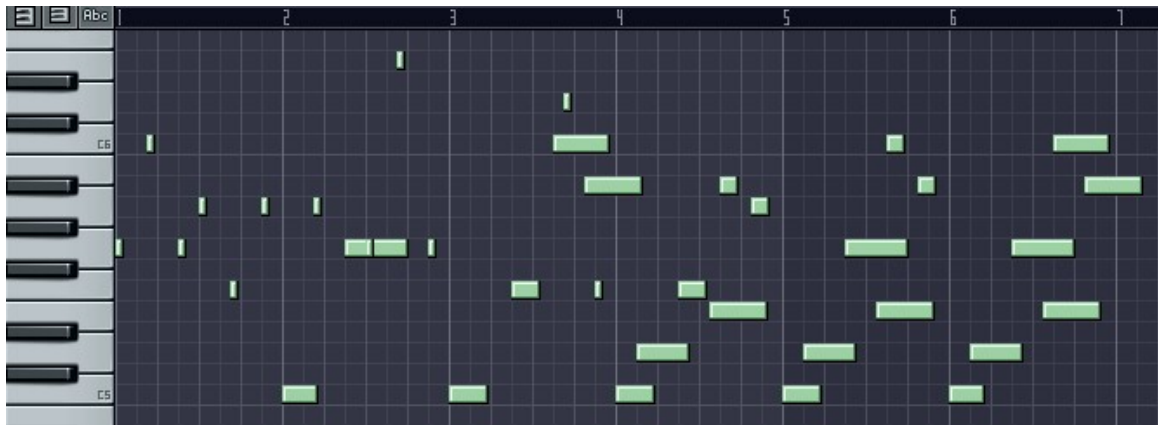


Figure 14 Example of a slower transform-speed yielding faster convergence. Settings are the same as previous example, except for the transform speed on *add/remove* which was reduced to 0.28.

In the **first** frame, as with the previous two examples, *scale pitch* applied a slight shrink, followed by an *inversion* that put the E5 at 3.75 up to E6. Unlike the previous morphs, *add/remove* replaced the F5 at 1.0 with C5 and the F5 at 2.5 with G5. This was entirely due to the difference in transform speed. In the previous example, patterns with the addition of E5, C6 and B^b5 were all judged to be equally close to the target dissimilarity of 0.1848 specified by the transform speed, whereas, in this example, the G5 was the only real candidate, fitting the target dissimilarity of 0.1974 very closely at 0.198. The next closest was the C5 at 0.1878 (this is observable in Appendix D).

In the **second** frame, a *divide/merge* merged forward, as with the previous example. However, unlike the previous morph, the E6 on 3.75 and the G5 on 4.5 was unaffected while the A5 at 1.75 was merged into the C5 at 1.0 and the G5 at 3.0 merged into the G5 at 2.5. Because of this, the subsequent *scale pitch* was milder, shifting all the notes down a tonal step except for the C5 on 1.0 which remained there because it is the tonic pitch of the most central octave and thus the centre of the scaling operation. The two cycles of *add/remove* resulted in the C6 at 3.5 and the B^b5 at 4.25.

In the **third** frame, the forward merge occurred again, as with the previous morph. Unlike the previous morph, the effect was to merge the F5 at 4.5 into the B^b5 at 4.25, rather than the G5 at 1.75 into the D5 at 1.7 (both of which did not exist in this case). Following this, *scale pitch* applied a slight shrink, shifting the C6 at 3.5 to B^b5 and the B^b5 at 4.25 down to A5. As a result, the

complete octave shift that occurred in the previous example was not needed. *Add/remove* added the D5 at 1.5 and the E5 at 3.25.

In the **fourth** and final frame, a slight pitch stretch was applied, shifting the B^b5 at 3.5 back up to C6 and the A5 at 4.25 back up to B^b5. Following this, only a single cycle of *add/remove* was required to replace the F5 at 2.5 with G5.

With only a slight change to the 'speed' parameter, from 0.46 to 0.28, this example is substantially different to the previous, thus highlighting the emergent nature of *TraSe*. The biggest difference occurred in frame three, with a slight pitch-shrink as opposed to a dramatic stretch. Compared to the previous morph, the overall structure at this point was maintained much more effectively and because of this, fewer frames were needed. This highlights some of the complexity in dealing with multiple transformations and dissimilarity measures, especially in terms of emergent flow-on effects.

Streamlining the transformations

A musical problem with *TraSe* is that mutation tends to be greater during the beginning of the morph, where multiple transformations are applied within a single frame, compared to the end of the morph, where the *add/remove* transformation performs slight adjustments. In response to this, the 'mutation limit' caps the number of transformations that may occur each frame. The effect is smoother changes during the beginning, however, more frames are generated.

This example ([~7.6](#)) has similar settings to the examples above, except the *add/remove* transform speed is set to 1.0 and the mutation limit is set to 2 transformations per frame.

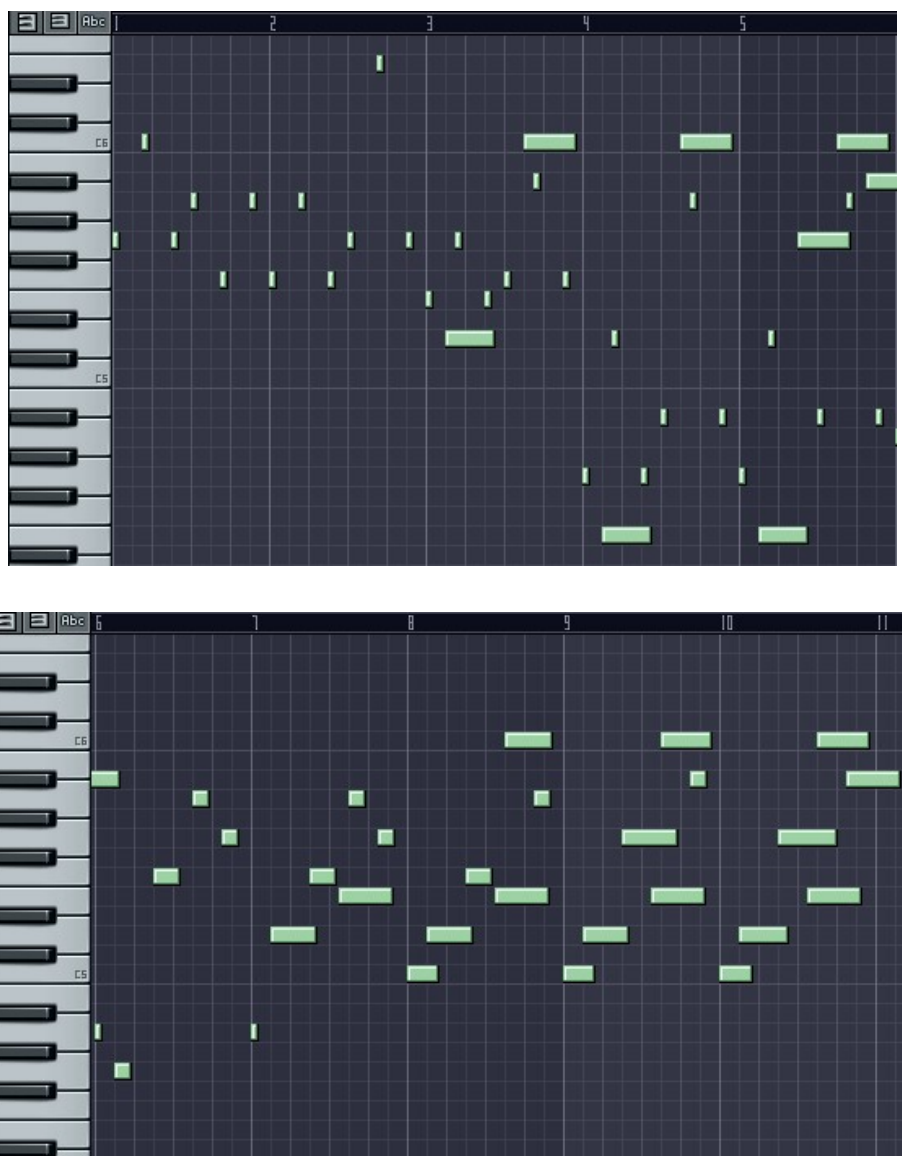


Figure 15 The number of transformations per frame has been limited to two. Merge is biased over split. *Add/remove* transform speed is 1.0.

In the **first** frame, a pitch-shrink and upwards inversion was applied, as with the first frame of the previous example, however, the limit of two transformations per frame obstructed the *add/remove* transformation entirely.

In the **second** frame, *scale pitch* shifted every pitch down by a tonal step, except for the E6 at 3.75 which was shifted down to B^b5. *Add/remove* added the D5 at 1.5 and the C6 at 3.5.

In frame **three** a dramatic *scale pitch* occurred, shifting the E5s at 1.0 and 2.5 to G5, the D5 at 1.5 to E5, the G5 at 1.75 up to D6, the F5s on 3.0 and 4.5 up to B^b5, the C6 at 3.5 up to C7, and

the B^b5 at 3.75 up to A6. The *octave* transformation then shifted the whole pattern down one octave.

In frame **four**, the only transformation applied was *add/remove*, which added the B^b5 at 4.25 and replaced the G4 at 2.5 with G5.

In frame **five**, a *divide/merge* applied a forward merge, consolidating the D5 at 1.75 into the E4 at 1.5, the B^b4 at 3.0 into the G5 at 2.5, the B^b5 at 3.75 into the C6 at 3.5 and the B^b4 at 4.5 into the B^b5 at 4.25. *Scale pitch* then applied a pitch-shrink, shifting the G4 at 1.0 up to A4, the E4 at 1.5 up to G4, the G5 at 2.5 down to F5, the C6 at 3.5 down to A5 and the B^b5 at 4.25 down to G5.

In frame **six**, *add/remove* was the only transformation applied, which added the E5 at 3.25 and replaced the G4 at 1.5 with D5.

In frame **seven**, *scale pitch* shifted the A5 at 3.5 up to B^b5 and the G5 at 4.25 up to A5. The first cycle of *add/remove* replaced the A4 at 1.0 with C5 and the second cycle replaced the B^b5 on 3.5 with C6.

In frame **eight**, *scale pitch* slightly shifted the C6 at 3.5 up to D6 and the A5 at 4.25 up to B^b5. The first cycle of *add/remove* reasserted the C6 at 3.5 and the second cycle replaced the F5 at 2.5 with G5.

From listening to this example, it is clear that imposing a limit on the number of transformations that occur each frame reduces the severity of mutation, particularly in the first few frames. The limit ensured only two transformations per frame, while the other examples above averaged three or more.

7.4.2 Morphing Backwards and Forwards

TraSe is currently a unidirectional morph, in that a morph generated from source to target will be different to a morph generated from target to source, even when using the same parameters and played in reverse. The first few frames result from large scale transformations while the last few frames are influenced mostly by *add/remove*.

In order to informally ‘cross-examine’ this effect, an example is included ([~7.7](#)) that has been generated from target to source (the same source and target as the previous section):

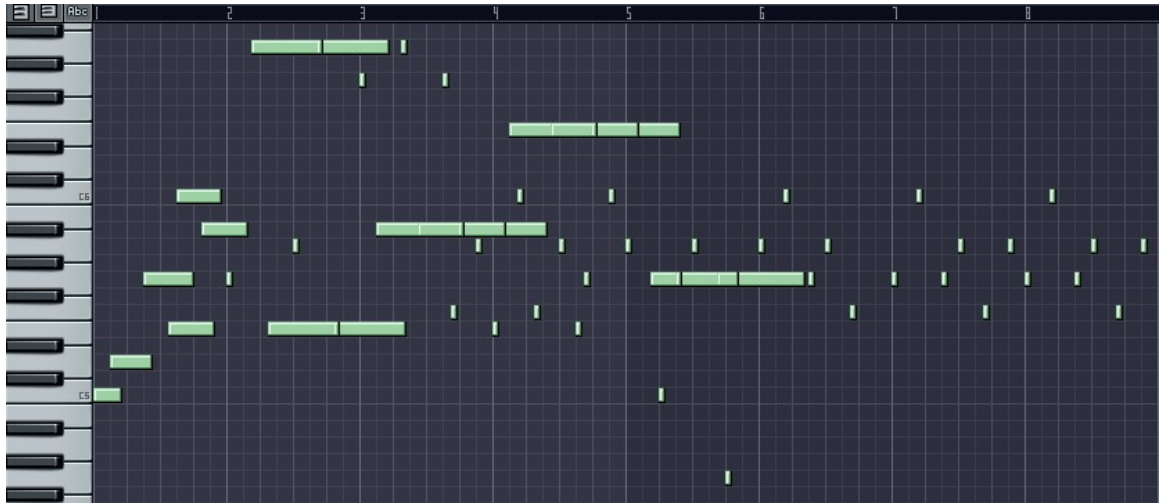


Figure 16 Morphing from target to source. The *add/remove* transform speed is set to 0.32 and merge is bias over divide. The dissimilarity cut-off is zero.

In the **first** frame, a *divide/merge* applied a forward-merge, which consolidated the D5 on 1.5 into the C5 on 1.0, the E5 at 3.25 into the G5 at 2.5 and the B^b5 at 4.25 into the C6 at 3.5. Following this, *rate* doubled the speed of the pattern. *Scale pitch* lifted the G5s on 1.75 and 3.75 up to A5 and the C6s on 2.25 and 4.25 up to E6. *Inversion* shifted the E6s on 2.25 and 4.25 down by two octaves to E4. *Octave* shifted the whole sequence up an octave. The first cycle of *add/remove* replaced the C5 at 1.0 with a staccato G5. The second cycle replaced the C5 at 3.0 with a staccato A5.

In the **second** frame, the forward-merge occurred again, this time consolidating the E5 at 2.25 into the A6 at 1.75 and the E5 at 4.25 into the A6 at 3.75. A slower *rate* increase to that of the previous frame was applied, shrinking the onsets by a factor of $\frac{2}{3}$ rather than $\frac{1}{2}$. The result brought the A6 forward to 1.5, the A5 at 3.0 to 2.33, the A6 at 3.75 to 2.83, with a repeat of G5 at 3.66, and A6 at 4.16. *Scale pitch* was then applied, shifting the A6s on 1.5, 3.75 and 4.16 to B^b6. An upwards *inversion* was then applied, shifting the A5 at 2.33 up two octaves to A7 and the G5 at 3.66 up two octaves to G7. The entire sequence was then shifted down an octave. The first cycle of *add/remove* added F5 at 3.75 while the second cycle added the F5 at 3.75.

In the **third** frame, *scale pitch* raised the G6s at 1.0 and 3.66 to E7, the B^b5s at 1.5, 2.83 and 4.16 up to E6, the A6 at 2.33 to F7, the F5 at 3.75 to G5 and the A5 at 4.5 to C6. This was followed by a severe upwards inversion, which shifted the E6s at 1.5, 2.83 and 4.16 up two octaves to E8, the G5 at 3.75 up two octaves to G7, the E6 on 4.16 up to E8 and the C6 at 4.5 to C8. This pattern was then shifted downwards by 2 whole octaves. The first cycle of *add/remove* added

the A5 at 3.0, while the second cycle added the C6 at 1.75.

In the **fourth** frame, the *rate* was reduced by a factor of $\frac{3}{2}$, shifting the E6 at 1.5 to 1.75, the C6 at 1.75 to 2.125, the F5 at 2.33 to 3.0, the E6 at 2.83 to 3.75, the A5 at 3.0 to 4.0; and remove the E5 at 3.6, G5 at 3.75, E6 at 4.16 and C6 at 4.5 altogether. This was followed by a severe *scale pitch* which shifted the E6 at 1.0 down to A4, the E6s at 1.75 and 3.75 up to G6, the F5 at 3.0 down to C5, and the A5 at 4.0 down to G5. An upwards *inversion* then shifted the A4 at 1.0 to A6 and the C5 at 3.0 to C7. The sequence was then dropped by a whole octave. The first cycle of *add/remove* replaced the C6 on 3.0 with an A5, while the second cycle added the G5 at 2.5.

In the **fifth** frame, a merge consolidated the C5 at 2.25 into the G5 at 1.75 and the G4 at 4.0 into the G5 at 3.75. No other transformations were applied on this frame except for *add/remove*. The first cycle replaced the G5 at 1.75 with C6, while the second cycle replaced the G5 at 3.75 with F5.

The **sixth** frame completed the morph with two cycles of *add/remove*, adding the A5 at 4.5 and replacing the A5 at 1.0 with G5.

Once again, this example demonstrates dramatic, yet fairly coherent, transformations during the beginning and middle of the morph, tapering away to very minor adjustments towards the end.

To provide another comparison, this morph was also played back in reverse order of frames, from target to source ([~7.8](#)):

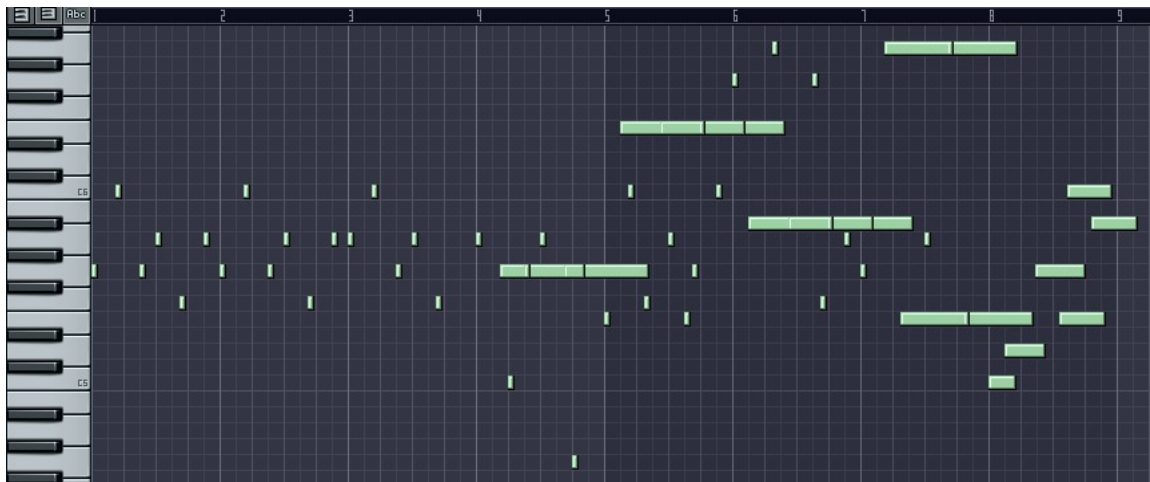


Figure 17 The same morph of the previous example, played in reverse order of frames.

The second half of this 'reverse' morph (Figure 17), appears to me to be somewhat less

coherent than the first half of the corresponding ‘forward’ morph (Figure 16), even though the frames are identical. This effect might be explained by the differing musical contexts and expectations of the two examples. Electronic music often consists of cycles of break-down and build-up, where the break-down occurs suddenly and the build-up occurs gradually. In the ‘forward’ morph, the quick cut away from the source can be considered analogous to a break-down, in that the continuity is broken. This also serves as a structural indicator, suggesting a new section. From the ‘pseudo-breakdown’, frames three, four and the target are a gradual ‘build-up’ of stability to the target sequence, thus fitting the standard pattern of break-down to build-up more closely. With the reverse order of frames, the contour of continuity is also reversed and this does not fit the standard ‘break-down, build-up’ structure. As well as this, there is no clear indication of the start of the morph.

Despite these interpretations, my subjectivity is biased from listening to more ‘forward’ than ‘reverse’ morphs over the period of this study. A formal test of ‘reverse’ and ‘forward’ morphs with multiple participants would be needed before any conclusive claims can be supported.

7.4.3 Key/Scale morphing with TraSe

The *TraSe* algorithm computes the morph for key and scale data separately to note sequence data. Examples of *key/scale* morphing are included here that cover modulating to a distant key and modulating through function change. A significant degree of flexibility and control is demonstrated by the *key/scale* morphing algorithm.

To recap the process: to generate a key/scale frame at each step, all the possible keys and scales are considered and rated according to their dissimilarity with the target. The dissimilarity rating is a weighted combination of *scale dissimilarity*, *key-scale dissimilarity* and *key-root distance*, which itself is a weighted combination of distances in the *CC* and the *CF*. Because all possibilities are considered, the key/scale transformation is almost a **perfect** transformation, which means that by adjusting the transform speed, the user can specify how many frames are generated.

Modulating to a distant key

Between C Major and F# Major only two of the pitch classes (F and B) are shared, as is evident in the table below:

	C Maj	F#
B		
A#		
A		
G#		
G		
F#		
F		
E		
D#		
D		
C#		
C		

Figure 18 Comparing the pitches in C Major with the pitches in F# Major

We will now examine select examples of morphing between C# and F# Major. Most of the examples below consist of 3 frames, which results from a transform speed of $\frac{1}{4}$.

If the *key-root distance*, *CC* and *scale dissimilarity* are weighted maximally, the expected result is a series of linear transpositions while maintaining the scale. This is confirmed with the following print-out:

```
frame 0 : root = 0 scale = ionian
frame 1 : root = 1 scale = ionian
frame 2 : root = 3 scale = ionian
frame 3 : root = 5 scale = ionian
frame 4 : root = 6 scale = ionian
```

Figure 19 Key/scale morphing between C Major (Root = 0, scale = Ionian) and F# Major (root = 6, scale = Ionian), using *CC*, *key-root distance* and *scale dissimilarity*.

If *CF* is used instead of *CC*, the transpositions will be intervals of a fifth:

```
frame 0 : root = 0 scale = ionian
frame 1 : root = 7 scale = ionian
frame 2 : root = 9 scale = ionian
frame 3 : root = 11 scale = ionian
frame 4 : root = 6 scale = ionian
```

Figure 20 Output from morphing between C Major (Root = 0, scale = Ionian) and F# Major (root = 6, scale = Ionian), using *CF*, *key-root distance* and *scale dissimilarity*.

If the *key-scale dissimilarity* is used exclusively, we obtain what at first appears at first to be an illogical result, ending on D# Aeolian, rather than F# Ionian:

```

frame 0 : root = 0 scale = ionian
frame 1 : root = 9 scale = mixolydian
frame 2 : root = 1 scale = aeolian
frame 3 : root = 11 scale = ionian
frame 4 : root = 3 scale = aeolian

```

Figure 21 Output from morphing between C Major (Root = 0, scale = Ionian) and F# Major (root = 6, scale = Ionian), using *key-scale dissimilarity* as the only measure.

On closer inspection, this is consistent with the specified task of finding the a sequence of key/scales that are related to each other by the pitch classes they contain; D# Minor (root = 3, scale = Aeolian) is the relative minor of F# Major and thus contains all of the same pitches. The logic of this progression is made clear in the following table:

	C Ioni	A Mixo	C# Aeol	B Ioni	D# Aeol
B					
A#				X	
A				X	
G#			X		
G			X		
F#		X			
F		X			X
E					X
D#			X		
D			X		
C#		X			
C		X			

Figure 22 The pitch-classes (rows) present in each key/scale frame (columns) during a morph from C Ionian to F# Ionian, unexpectedly finishing on D# Aeolian. The “X” marks where there is a difference between the previous scale and the scale with the X, in terms of the pitch-class that the X is on.

In order to converge with the specified target, a very small fraction of *key-root distance* can be included:

```

frame 0 : root = 0 scale = ionian
frame 1 : root = 6 scale = phrygian
frame 2 : root = 6 scale = dorian
frame 3 : root = 6 scale = mixolydian
frame 4 : root = 6 scale = ionian

```

Figure 23 Output from morphing between C Major (Root = 0, scale = Ionian) and F# Major (root = 6, scale = Ionian), using absolute pitch-class similarity combined with a small (0.01) weighting of key-root distance.

Compared with the previous example, this contains exactly the same set of pitches, but ends on the specified target of F# Ionian:

	C Ioni	F# Phry	F# Dori	F# Mixo	F# Ioni
B					
A#				X	
A				X	
G#			X		
G			X		
F#		X			
F		X			X
E					X
D#			X		
D			X		
C#		X			
C		X			

Figure 24 The pitch-classes (rows) present in each key/scale during a morph from C Ionian to F# Ionian using absolute pitch-class similarity combined with a small (0.01) weighting of key-root distance. The “X” marks where there is a difference between the previous scale and the scale with the X, in terms of the pitch-class that the X is on.

It is interesting to note that the pitch class of F is absent during the morph, even though it is present in both the source and the target. This demonstrates how compositional transformations indirectly influence the musicality of the morph. Also worth noting is the immediate shift from C to F#. Musically it may be more effective for this shift to occur half-way through the transition, however, currently it is not possible to control this aspect. Given options with the same *key-scale dissimilarity*, the algorithm will select the key of the target.

Modulating though function change

When analysed in terms of harmonic function, the key/scale morph can demonstrate some common forms of modulation. For this, a source of C-Ionian and target of B^b-Ionian has been chosen due to similarity and common usage, rather than dissimilarity and difficulty (see previous examples for this).

With *key-scale dissimilarity* weighted 1.0, *key-root distance* weighted 0.5, *CF* weighted maximally over *CC* and *scale distance* weighted 0, a I-IV-ii-V-I progression is obtained where the I-IV is relative to the source key and the ii-V-I is relative to the target key:

```
frame 0 : root = 0 scale = ionian
frame 1 : root = 5 scale = lydian
frame 2 : root = 0 scale = dorian
frame 3 : root = 5 scale = mixolydian
frame 4 : root = 10 scale = ionian
```

Figure 25 Modulating from C Ionian to B^b Ionian. *Key-scale dissimilarity* is 1.0, *key-root distance* 0.5, *CF* 1.0, *CC* 0.0, *scale distance* 0.0, *track VS consistency* 0.37 and *transform speed* 0.25.

This kind of ii-V-I progression is commonly cited by jazz musicians, for example see (Pachet 1997; Pachet 1999).

The *key/scale morph* can also reveal coherent modulation pathways that are not particularly obvious. For example, the following progression of I-vii-ii-I, where the vii-ii-I is in the key of the target. This is arguably a coherent, if a little edgy, progression:

```
frame 0 : root = 0 scale = ionian
frame 1 : root = 9 scale = lochrian
frame 2 : root = 0 scale = dorian
frame 3 : root = 10 scale = ionian
```

Figure 26 Modulating from C Ionian to B^b Ionian. *Key-scale dissimilarity* is 1.0, *key-root distance* 0.66, *CF* 1.0, *CC* 0.66, *scale distance* 0.0, *tracking VS consistency* 0.0 and *transform speed* 0.25.

When the “track VS consistency parameter” is increased to the upper limit (0.77 for this example) a smoother progression is generated, because the measure will include dissimilarity with the source, rather than purely the target. This is I-V-ii-I, where the V is in the key of F (5) and ii-I is in the key of the target, B^b(10):

```

frame 0 : root = 0 scale = ionian
frame 1 : root = 0 scale = mixolydian
frame 2 : root = 0 scale = dorian
frame 3 : root = 10 scale = ionian

```

Figure 27 **Modulating from C Ionian to B^b Ionian.** *Key-scale dissimilarity* is 1.0, *key-root distance* 0.66, *CF* 1.0, *CC* 0.66, *scale distance* 0.0, *tracking VS consistency* 0.77 and *transform speed* 0.25.

A slow transform speed can be applied to generate extended chord progressions. In the following example, a transform speed of 0.08 yields: I-ii-V-ii-IV-V-I where I is in the key of C, the following ii is in F, the vii-V-ii is in the target key of B^b, the IV is in F, the last V is in E^b before concluding with the I of B^b:

```

frame 0 : root = 0 scale = ionian
frame 1 : root = 7 scale = dorian
frame 2 : root = 9 scale = lochrian
frame 3 : root = 5 scale = mixolydian
frame 4 : root = 0 scale = dorian
frame 5 : root = 10 scale = lydian
frame 6 : root = 10 scale = mixolydian
frame 7 : root = 10 scale = ionian

```

Figure 28 **Modulating from C Ionian to B^b Ionian.** *Key-scale similarity* is 1.0, *key-root distance* 0.66, *CF* 1.0, *CC* 0.66, *scale distance* 0.0, *tracking VS consistency* 0.0 and *transform speed* 0.08.

In summary, *key/scale morphing* is an effective application of *TraSe* because the entire search space is known, thus affording control over the number of frames generated. Flexibility has also been demonstrated, for example, through a range of solutions to the ‘difficult’ modulation from tonic to tritone. With the ‘easier’ modulation task of stepping down a whole tone, well known progressions such as ii-V-I were able to be generated.

7.4.4 When Johnny comes morphing home

Morphing between long, continuous melodies is difficult, because the large scale transformations are unable to perform the small-scale modifications that are necessary. This is evident in the following example from *The British Grenadiers* and *When Johnny Comes Marching Home*. This morph relies almost exclusively on the *add/remove* transformation, the other transformations being unable to offer any suitable modifications. The inadequacies of this are obvious in this example ([~7.9](#)). While this clearly demonstrates some problems, it is rare for extended tunes to occur in the short loops of MEM that are the research context. Nonetheless, the need for phrase analysis as a preliminary step to *TraSe* morphing is clear, as mentioned in more detail below (7.7.2).

7.5 Automated evaluation

Automatic evaluation of *TraSe* involved programmatically generating a statistically significant number of source and target test examples and, with default parameter settings, generating morphs for them. This technique was used to verify the time complexity and to investigate the correlation between the number of frames generated and the number of notes in the source and target note sequences. The time complexity has relevance to realtime operation, while the correlation between the number of notes in source and target and the number of frames generated is relevant to the issue of over-generation of frames. For these tests, the results using the whole transform-chain were compared to the results using only *add/remove* as a base-level comparison.

7.5.1 Computation time for a single frame

The time complexity of *TraSe* is $O(n^3)$, due to the *add/remove* transformation and the NN dissimilarity measure. In order to confirm this empirically, the time it takes to compute one cycle of *add/remove* for a single frame has been measured for different numbers of notes. No bias weightings were applied and the notes were equally distributed between the source and target.

The note sequences were generated randomly, with pitches ranging between 40-80 and onsets ranging from the start to the end of the loop. Velocity and duration was constant. The computation time was measured for 4 notes, then again with 54 notes, and so on up to 204 notes, adding 25 notes to source and target each time. The time was measured three times on each occasion to confirm the accuracy of measurements:

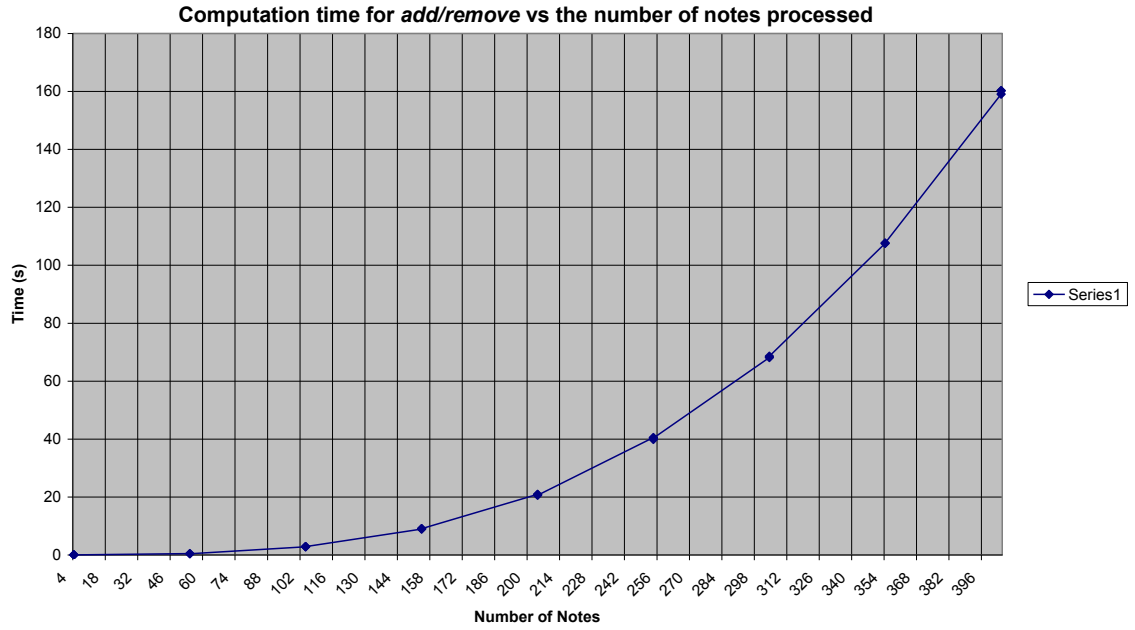


Figure 29 The computation time (y) for a single cycle of un-biased *Add/remove*, applied to various numbers of notes (x), which were randomly generated and evenly distributed between source and target.

These measurements were then fitted to polynomials from degree 1 to degree 4 in order to determine whether this curve was the result of $O(n)$, $O(n^2)$, $O(n^3)$ or higher. It was found that the first and second degree polynomials could not sufficiently describe the curve, while polynomials of third degree and above fit it perfectly, providing empirical evidence that the complexity of *add/remove* is $O(n^3)$ (see Appendix D, D-1).

While this justifies predictions concerning the time complexity of *add/remove*, less consistent results were discovered when the entire transformation chain was applied to the same randomly generated sequences:

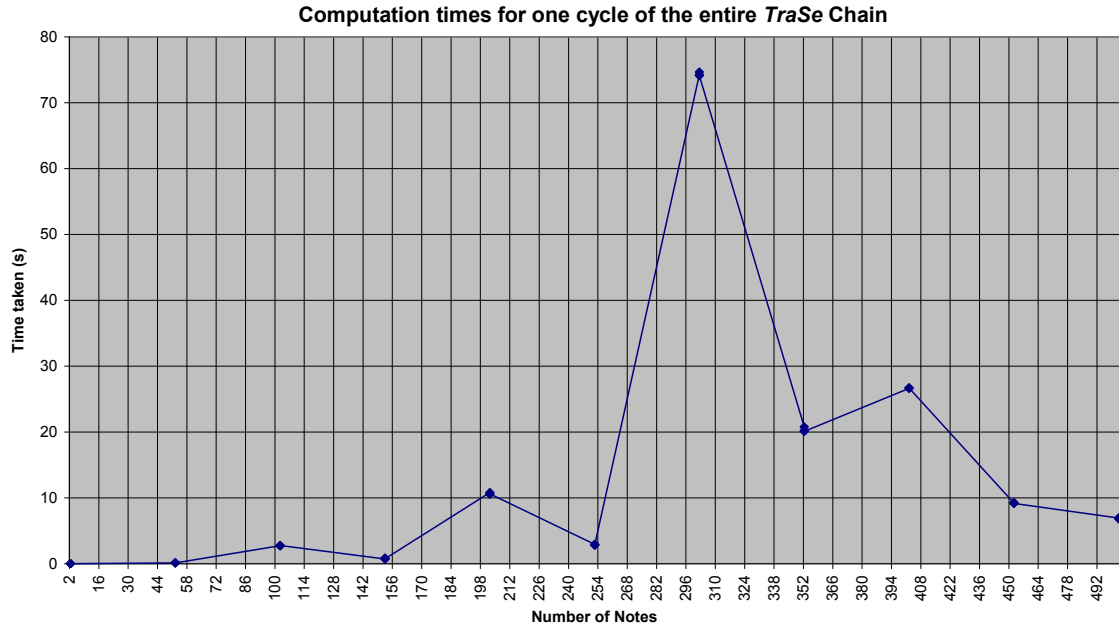


Figure 30 The computation time (y) for a single cycle of the un-biased transform-chain, applied to various numbers of notes (x), which were randomly generated and evenly distributed between source and target.

First of all, it is clear from the peak time of 80 seconds, compared to the 160 seconds of the previous example, that a single cycle of the transform-chain will usually take less time to compute than *add/remove* only. Another difference is that, although the computation time increases with the number of notes, there is a huge amount of variability.

This can be attributed to the emergent flow-on effects of *TraSe*, due to the variety of large-scale transformations. While predictions can be made regarding the critical aspects of complexity, the efficiency experienced during execution of *TraSe* will usually deviate substantially. This indicates a chaos-like susceptibility to initial conditions.

It should be noted that the data obtained above (Figure 29 and Figure 30) was for only a single frame of the morph – various combinations of music and transformation parameter configurations will require different numbers of cycles before convergence is reached, which also has an effect on the time complexity.

7.5.2 Number of notes and number of frames generated

The computation time of *TraSe* clearly increases with the number of frames generated. As well as this, the coherence of the morph can be adversely affected. For example, with many frames,

the default operation is for only small snippets from each frame to be played and it can become impossible to hear the logic of the entire progression. The automated tests presented below have been selected so as to best demonstrate the correlation between the number of frames generated and the number of notes in the source and target.

As a baseline for comparison, the first example shown below uses only *add/remove* to generate the morph. The randomly generated material is monophonic and quantised to 0.25 intervals within a single bar. To minimise the extent by which identical notes are generated in source and target, and thus to further challenge the algorithm, the pitches are generated in different octaves: between C5-C6 in the source and between C6-C7 in the target. The *add/remove* transformation was set to 2 cycles, monophonic mode and the transform speed was 1.0. There were 50 sample morphs randomly generated for each number of notes from 1 to 16 (equally in both the source and target):

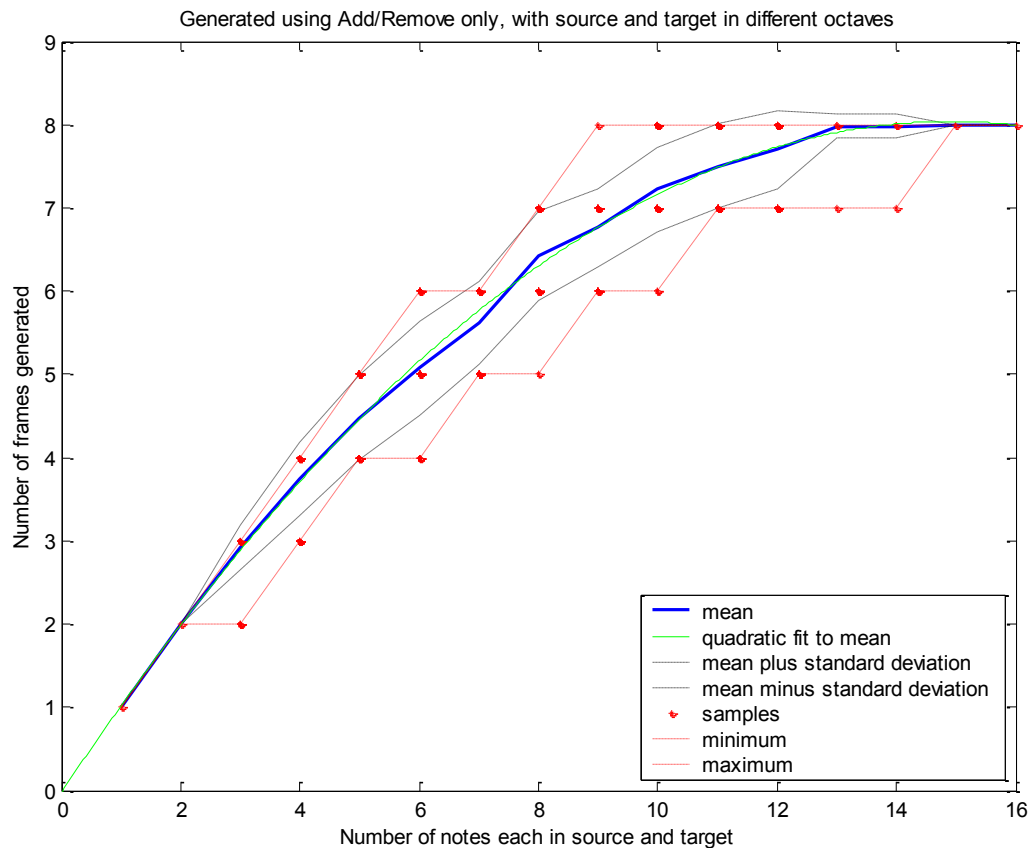


Figure 31 Number of frames generated with two cycles of *add/remove* only, monophonic, quantised to 0.25, source constrained to C4 octave and target constrained to C5. For each number of notes there were 50 samples.

The logarithmic curve can be explained by the way *add/remove* functions in monophonic mode, replacing notes in a single operation if they are on the same onset as the note being added. With sixteen notes in both source and target, quantisation at 0.25 and the loop length of 4 beats, each cycle of *add/remove* will be twice as fast than if the notes had to be removed and added separately.

When the entire transform-chain is applied to source and target material that is generated in the same way – that is, in separate octaves so as to eliminate the possibility of exactly matching notes in source and target– many more frames are generated, on the whole:

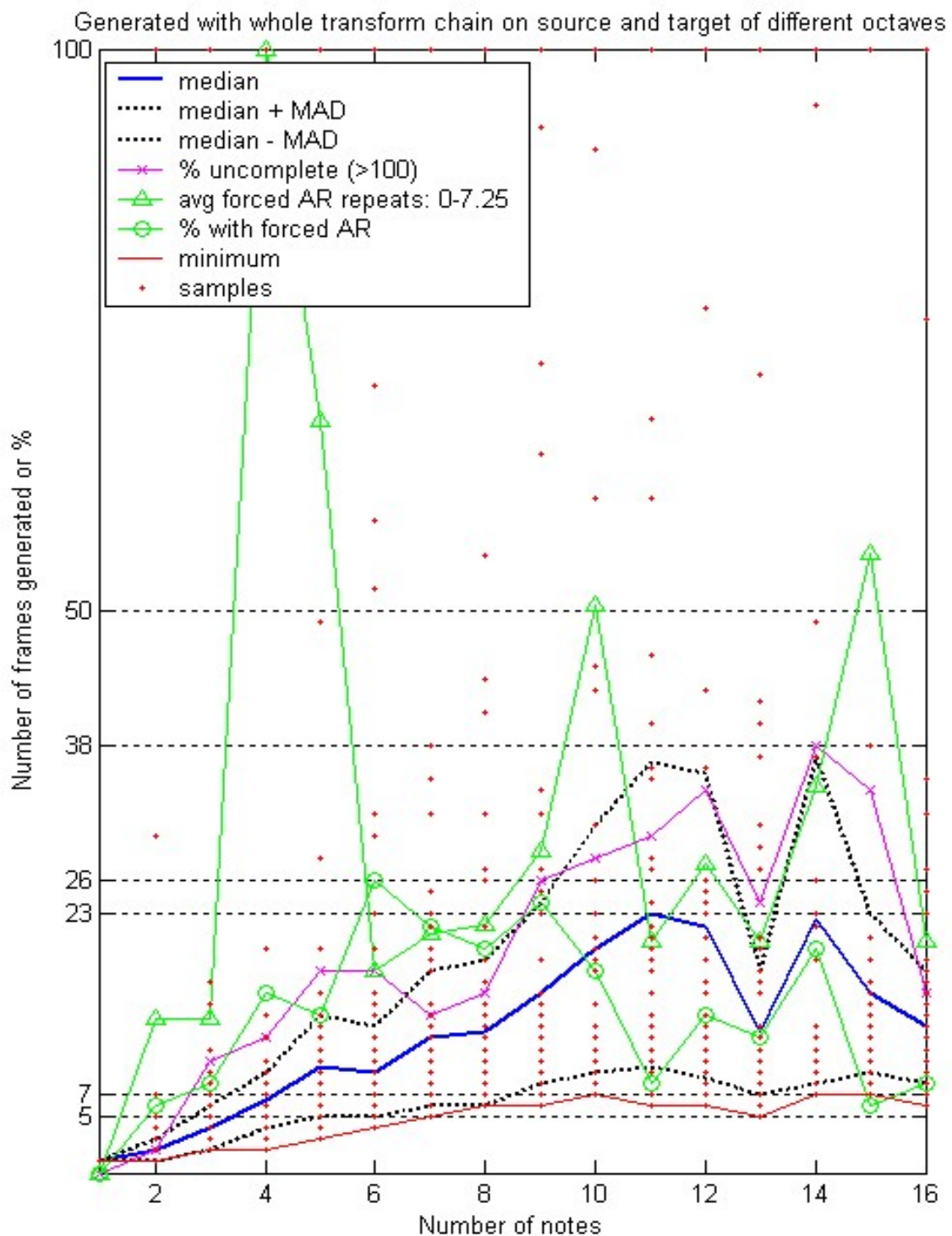


Figure 32 Number of frames generated using the entire transform-chain, versus the total number of notes in the source and target patterns. Both source and target are monophonic and quantised to 0.25. Source is constrained to C4 octave and target constrained to C5. MAD is 'Median Absolute Deviation'. AR is 'Add/Remove'.

However, generating morphs with the transform-chain outperforms *add/remove* by itself in terms of minimum number of frames produced in each sample group. The comparison is made clear below:

<i>add/remove</i> only	1	2	2	3	4	4	5	5	6	6	7	7	7	7	8	8
All trans.	1	1	2	2	3	4	5	6	6	7	6	6	5	7	7	6

Figure 33 With source and target being generated on separate octaves, comparing the minimum number of frames produced at each number of notes when using “*add/remove* only” VS “all transformations”. The yellow highlighted columns refer to occurrences where the minimum “all transformations” was smaller than the minimum of “*add/remove* only”.

Some of the gains in terms of minimum frames generated could be attributed to the “octave shift” transformation, which would shift the entire source into the same octave as the target, thereby increasing the chance of notes being the same. As well as this, a source and target combination will occasionally occur where the large-scale effects of the compositional transformations happen to produce a rapid mutation into the target. It is unlikely to be purely random considering the already large sample size of 50. In addition to this, the extent of the difference is significant, with the ratio of smaller minimums for “*add/remove* only” : “all transformations” being 2:8.

These automatically generated examples demonstrate the difficulties that *TraSe* has without manual parametric control from the user. In particular, the problem of over generation of frames remains evident, with the morphs that could not converge after 100 frames peaking for 14 notes at 38% of the samples. Despite this, the potential of compositional transformations to assist convergence in fewer frames than *add/remove* has also been illustrated by comparing the minimum number of frames generated in each.

7.5.3 Future improvements to automatic testing

A number of possible improvements to the automatic testing procedure have become evident. Another test that is likely to show interesting results would be to apply the entire transform-chain on ‘musical’ loops, compared to randomly generated loops within the same general constraints. Also, to support comparisons, the same material should be applied in each test, rather than generating fresh material each time. Another useful set of tests would be on source and target with different numbers of notes, rather than evenly distributed between them. Although larger-

scale tests — such as applying the entire transform-chain to large sample sizes, un-quantised material and pitch ranges greater than an octave — could be executed over many days or on a super-computer, the main problem of too many frames before convergence, as evidenced in the current tests, will still remain and no solutions are likely to emerge. The research effort could be better spent on implementing some improvements to the current algorithm that are explained in 7.7.

7.6 Formal qualitative evaluation: web questionnaire

A web questionnaire was developed as an empirical qualitative musicological investigation into morphing between MEM loops of various styles. The two primary aims were:

1. Collect knowledge that can be used to improve future morphing algorithms.
2. Establish if *LEMorpheus* is likely to be successfully applied to real-world contexts.

These were broken down further into more specific objectives:

1. Discover elements of *LEMorpheus* generated morphs that are perceived to have a positive or negative impact, benchmarking against human composed morphs.
2. Obtain techniques for morphing.
3. Determine if there is a correlation between smoothness and effectiveness. Smoothness is defined as moment to moment continuity, whereas effectiveness is defined as the ability the music to affect the listener in a way that is perceived to be the intention of the composer.
4. Obtain opinions as to whether the morphing techniques would be applicable to games or live electronic music performance.

The approach was to research, compose and recreate sets of MEM loops, create *LEMorpheus* morphs between them and pay for a professional music producer to compose and produce morphs manually. The questionnaire was developed according to the objectives, widely publicised and completed by nine participants with musical backgrounds. Australian respondents were paid and feedback was collated and analysed.

In terms of results, *LEMorpheus* morphs were characterised by progressive musical variations, while the human composed morphs utilised layering and larger structural changes. While the majority of respondents favoured the human composed music, there was a great deal of

controversy in opinion, due to participants attending more or less to different elements in the music and differences in musical background.

A number of techniques for morphing were gathered, including the blending of similar elements, removal of dissimilar elements, trading of loops (as with 'beat juggling'), incremental substitution of phrases, gauging of source and target dissimilarity to determine whether to blend or switch directly, generation of a separate bridge section or breakdown and synchronisation of rhythm and evolution of other elements.

There was found to be very little correlation between smoothness and effectiveness. For example, many human composed morphs were found to be coherent but not smooth, while many *LEMorpheus* morphs were smooth but not coherent. Some terminological ambiguity with 'smoothness' was apparent in some responses, however, there was sufficient additional explanation for this not to impact significantly on the validity of the data.

Overall, *LEMorpheus* was rated, near-unanimously, as being applicable to both computer games and live electronic music contexts. The few instances of a negative result was balanced by similar reactions to the human composed morphs.

7.6.1 Method: music creation

I composed the four sets of source and target loops, generated morphs for them using *LEMorpheus* and hired a composer/producer to compose "benchmark" morphs. The source and target music were not based on any particular track, but were creatively composed, so that a high level of dissimilarity could be 'designed' into the examples. The morphs were generated by iteratively adjusting *LEMorpheus* parameters and auditioning the results. The benchmark morphs were composed through a reflexive process of listening and editing.

All of the musical examples used in the questionnaire are included:

Example One

Source ([~7.10](#)). Target ([~7.11](#)). LEMorpheus morph ([~7.12](#)). Composer morph ([~7.13](#)).

Example Two

Source ([~7.14](#)). Target ([~7.15](#)). LEMorpheus morph ([~7.16](#)). Composer morph ([~7.17](#)).

Example Three

Source ([~7.18](#)). Target ([~7.19](#)). LEMorpheus morph ([~7.20](#)). Composer morph ([~7.21](#)).

Example Four

Source ([~7.22](#)). Target ([~7.23](#)). LEMorpheus morph ([~7.24](#)). Composer morph ([~7.25](#)).

To compose each example, I applied my own composition skills and familiarity with the genre of MEM. The loops were designed to contrast drastically in ways that included: timbre, metre, rhythmic pattern, note density, key, scale, dynamics and phrasing. The parts were limited to drums, bass, lead, chords and auxiliary sounds/percussion. Occasionally, parts that were too long were split into two different layers (as if they had been clustered into two different phrases) to enable less frames to be generated. The music was also subjected to informal criticism by peers. I used the *LEMorpheus* sequencer which drove a custom developed patch in the *ReasonTM* synthesiser. The first three examples were composed with contrasting timbres to reflect real-world application contexts. The fourth example had no timbral difference and was included to compare the influence of timbre in the analysis of the note sequence morphing techniques of *TraSe*.

The *LEMorpheus* morphs were sixteen bars long. There were different parameter settings for each part and these deviated to some extent from morph to morph. For drums, harmonic transformations such as pitch stretch, harmonise and octave shift were bypassed. For other tonal parts, one or two transformations were bypassed when they caused over-generation of frames. The number of cycles of *add/remove* was generally kept low, at two or three, although, occasionally this had to be increased up to seven. There were usually around ten frames, which is an adequate fit considering sixteen bars, however, some parts had more, for example, the drums of the second morph with twenty-five frames. The morph examples usually had three to four *key/scale* frames which served to divide the morph into distinct sections. Fewer than this appeared to break the continuity, while more than this lost the sense of tonality. *TraSe* was used almost exclusively, however, a 'straight switch' was used for some of the cymbals/auxiliary parts,

and the *Markov Morph* was used for one layer of the lead in the second example.

Ande Foster (2007) was hired to produce the benchmark examples. The task was to morph smoothly from source to target while maintaining coherence. The limit was also sixteen bars, however, in example two, some flexibility in the length was allowed for the creative metrical integration of $\frac{6}{8}$ and $\frac{4}{4}$. Foster was provided with MIDI files of each source and target and a Reason™ patch for each part. No other musical materials, for example loops from a sample library, were allowed. The composer was encouraged to avoid recognisable sequences and compose creatively with the resources given. From the video documentation “talk through” of his compositional process, some observations can be made regarding Foster’s approach:

- Establishing rhythmic coherence is usually the primary task, followed by designing a coherent structure and integrating melody and harmony.
- If there is a way to play the music together with little modification, this is preferred.
- Ambiguous tonality can be employed effectively during transitions, especially in difficult key changes.
- Different timbres in source and target afford more drastic changes in the music, such as breakdown. As well as this, parts are ended (either faded out or stopped dead) earlier than in the current morphing algorithms and at musically significant points.
- When timbres are the same, ‘smoother’ processes that are similar to the techniques of morphing are preferred.

In summary, the music creation consisted of three stages: composition of source and target music, generation of morphs using *LEMorpheus*, composition of benchmark morphs. The source and target music was composed by myself, designing the sequences to be dissimilar. The *LEMorpheus* morphs were generated through adjusting parameters. The benchmark morphs were commissioned by a professional composer/producer.

7.6.2 Method: questionnaire

The questionnaire was designed to discover more about the positive and negative aspects of the morphs, new techniques, any correlation between smoothness and effectiveness and the real-world applicability of the morphs. For most of these, qualitative musicological analysis was chosen as the most fitting approach to obtaining such complex information.

The questionnaire, as implemented in flash, can be viewed ([~7.26](#)), the '.swf' file can be run from a flash enabled browser. Source files and the XML with raw results of the questionnaire are available in the accompanying Rom, in the folder '7. digital Appendix'.

The questionnaire had four sections for each of the four musical examples and a page at the end to ascertain the musical background of the respondent, record the completion time and elicit feedback. Each of first four sections used the same eight pages but with different musical examples. The first two pages related to the source and target material and the following six for the two morph examples – three pages each of the same questions. Each musical example could be played, paused and shifted to any point within the music, with the time displayed in seconds. The bias from multiple listening that this would introduce was not considered a problem, as it is much easier to perform musicological analysis with repeated listening. As well as this, in many applications, the music would be played multiple times (computer games, in particular).

Questions on source and target

The first page elicited a 'subjective response' to the source and target. A list explaining various attributes that might be used was provided:

- Intensity (strong VS weak, or in-between).
- Quality (positive VS negative, for example, happy, uplifting or feel-good VS sad, dark or depressing, or in-between).
- Recognisable parts, phrase groups within parts, and relationships between them.
- Connections between what you hear and music you have already heard: musical references, stylistic traits and their implications.
- Whether it sounds 'right', like every aspect of the music had a relevant purpose or role within the style, or 'wrong', like the composer/producer made a mistake and didn't bother to fix it up.

Figure 34 **List of stimuli for subjective response.**

The focus of the second page was on morphing techniques. Respondents were asked what their approach and techniques would be and how the elements of the source and target music would affect the task. A list of musical elements, adapted from Pratt (1998), was provided:

- Metre-underlying pulse; cycle lengths, regular or varying.
- Rhythms-uniform or non-uniform.
- Pitch tonality-pentatonic (or fewer notes); seven notes, major, minor or modal; tonal or atonal.
- Texture - monophonic, independent lines of counterpoint or homophonic chords.
- Timbre - different sounding instruments may serve a similar musical function in source and target.
- Range - the range of pitches covered by the different parts.
- Density - the number and distribution of notes.
- Dynamics - fluctuations in loudness and softness of the instruments.
- Articulation - the degree of accent/attack and separation/release of notes.
- Pace - how frequently any or all of the elements above occur and change.
- Structure - what kind of phrasing or logical structure is behind the organisation of musical events.

Figure 35 **List of musical elements to assist musicological analysis.**

Questions on morphing examples

On the first page for the morph examples, a subjective response from the listener was required, with the list in Figure 34 being repeated for stimulus. The respondent was asked to rate the morph according to how smooth and continuous it was, with values from one to seven. They were then asked what affect the smoothness had on their subjective response. Lastly, they were asked whether the changes had been effectively applied, even if the morph was discontinuous; or, if the morph was smooth, whether the smoothness made it effective.

For the second morph-related page, the respondents were asked to listen to the music and identify four important changes, both musically acceptable and not. They were asked to record the time (in seconds) of the changes, their subjective reaction to it and a musicological justification of the subjective reaction. The list in Figure 35 was included for stimulus.

The third morph-related page was focused on real-world applications. The first question asked whether the morph would be acceptable in a real-world context such as a computer game or live electronic music show. The second question asked the respondent for small adjustments they would make to the morph if they were a composer/producer and their task was to fix the morph in five minutes. The time limit was imposed so that efficiency was considered in combination with the ideal result. They were then asked for changes they would implement if more time was available.

Background

The final page of the questionnaire asked the participants about their musical background, including music they have been exposed to, educated about and compose/perform. This was placed at the end of the questionnaire so the answers given were not augmented by the categories the respondents placed themselves under. They were also asked to comment on the questionnaire and estimate how much time it took them to complete it.

7.6.3 Results and analysis

The questionnaire was completed by nine participants⁵, each taking two hours on average. There was a consistent qualitative response to most of the source and target material which highlighted the dissimilarities of source and target examples. The approach to morphing proposed by most respondents consisted of similarity/difference analysis, thinning of dissimilar elements and rapid switching/cutting. A range of specific compositional techniques were also proposed. The source and target music had little influence on the techniques.

Subjective responses to the morphs varied considerably, depending on which aspect of the music attention was paid to. The typical result was a moderately negative response to the *LEMorpheus* morph, and a moderately positive response to the transition which was hand-composed by the professional; however there were some notable exceptions.

Smoothness was not correlated to effectiveness, in fact, if the source and target music was very dissimilar, most participants advocated that a smooth transition should not be attempted. Some respondents were confused by the notion of smoothness; although, the rich qualitative

⁵ More than sixty people began the questionnaire but did not complete it

information from various questions enabled a sufficient level of triangulation to interpret their response.

Obvious changes were usually noted as being important to all respondents, but would sometimes invoke different aesthetic responses. Other changes were highlighted by some respondents and not others. Changes noticed in the *LEMorpheus* morphs were usually musical variations, while changes noticed in the human composed morphs were usually layering or structural changes. This is due to the fact that the *LEMorpheus* algorithms do not utilise any techniques of structural form other than continuity.

On the whole, both the human composed and *LEMorpheus* generated morphs were judged to be applicable to real-world contexts, but a few respondents judged particular examples otherwise, both for *LEMorpheus* and the benchmark. All but one participant was male, and their backgrounds were mostly all in music composition, production or performance over a range of genres. One participant was an avid music listener but did not otherwise have a formal or professional musical background. The amount of experience ranged from three years to thirty-five.

The background of each participant is summarised below:

Participant	Background
One	30 years. Classical to pop, art, electronic. Researching/teaching. Male.
Two	30 years. Classical to pop. Researching/teaching. Male.
Three	15 years. Classical to pop, indie, electronic. Producing. Male.
Four	40 years. Electronic, art, pop. Teaching. Male.
Five	15 years. Alternative pop to electronic. Listening. Male
Six	15 years. Alternative pop to electronic. Researching/Teaching. Male.
Seven	4 years. Electronic, pop. Producing. Male.
Eight	20 years. Classical to art/electronic. Researching. Female.
Nine	10 years. Alternative pop to classical to electronic. Researching. Male.

Figure 36 Summary of participants' backgrounds.

Subjective responses to source and target

The subjective responses to source and target music from most respondents have been summarised below (Figure 37). Because a range of different terms were used, a degree of interpretation was involved:

	Example 1	Example 2	Example 3	Example 4
Source	Dark and intense.	Slow, jazzy.	Ambiguous, sunny.	Upbeat Latin/African.
Target	Happy, party.	Upbeat funky.	Horror, intense.	Straight disco.

Figure 37 Summary of perceptions of the source and target material.

Despite this general trend, there were a number of anomalies. For example, participants two and seven perceived the source of example one to be “confused” and “jarring” in an unintentional way, partly because of the lack of a surrounding context within which to situate the loop. Despite this, it was still clear that the ‘dark/intense’ mood had still been perceived. Contrastingly, participant five perceived it to be “nice” and “trancey” and participant eight said “disco”, “good sound” and “dance”, without providing a mood indicator. Participant eight perceived the target of example two to be “too fast, and high pitched”. Participant seven identified the mood of example two source, but perceived it to be “not a good look... too slow to be successful”, although this was qualified by “not really my kind of thing”. A similar response was elicited from participant three. The most disliked loop of all was the source of example three. Despite this, only participants six and seven doubted the credibility of the music. Recall that the source and target of example four had contrasting musical elements, but with the same timbre. Despite the mostly mild responses, some minor criticisms of the example four source music were raised, with comments on note density, repetition and timbre selection.

Composition Techniques

A diverse range of compositional approaches were evident. Five participants suggested blending compatible elements and removing incompatible elements. Three participants advocated that the dissimilarity between the source and target should be gauged before deciding whether to attempt blending (if similar) or rapid switching (if dissimilar).

Rhythm was often mentioned before other elements and two participants explicitly stated that synchronisation of rhythm should occur first, followed by evolution of timbre and harmony. The various approaches to the task and the participants that proposed them have been summarised in the table below. Some degree of interpretation was required to condense the responses into this summary.

Participant (s)	Overall approach
1,9	Break into phrases and tracks. Do substitutions, one at a time, while maintaining a consistent thread.
3	Remove incompatible elements trade loops, like juggling tracks on a turntable.
3, 4, 9, 7	Gauge distance, either blend or switch depending on distance.
4	Write bridge with similar materials. Blend into and out of the bridge.
5, 7, 2, 9, 6	Blend particular elements that are easy to blend, remove elements that are difficult.
8	Evolve rhythm and timbre while cross-fading.
2, 6	Strip back, or use a breakdown in middle
5, 6	Synchronise rhythm first, then evolve other elements.

Figure 38 Summary of overall approaches used by participants.

The most striking feature of the responses to this section of the questionnaire was the diversity of specific techniques considered. The range of responses has been collated and are presented in Appendix E, Figure E-1 and E-2.

Smoothness and effectiveness of the morphs

The judgements of smoothness and effectiveness were mixed, however, most participants did not correlate the two. The human composed morphs were mostly considered to be the more effective, but also the least smooth. The *LEMorpheus* morphs were typically the opposite, however, there were notable exceptions to this, particularly in examples two and four, where the comments regarding effectiveness were evenly divided.

Some confusion regarding the definition of smoothness was evident for some participants and two different notions of smoothness emerged:

1. Continuity in mood/feel, so that subtle changes occur regularly throughout the morph, each one being a mild augmentation that maintains some aspect of mood or feel. (the expected definition)
2. Continuity in the level of satisfaction within the listener, as brought about by expectation and fulfilment. In this case, a dramatic change can be 'smooth' if it is somewhat predictable, well-timed and framed appropriately. (an unexpected definition which confounds appreciation with smoothness).

Most participants adopted (1), for example, participant seven on the human morph of the first example:

"I have only rated this morph as 5 on smoothness- but I would give it 7 for effectiveness- the drop into the new beat is not really smooth- but it's perfect. So smoothness is only required when appropriate."

Other participants adopted (2), for example, participant two on the human composed morph of the first example:

"... more conscious focus on establishment than too many changes... works better for me, seemed smoother."

Sometimes the understanding of smoothness was blurred. For example, when participant five described a change he would make to the human morph of example one:

"I would drop the sequence from approx 18 seconds through to 21 seconds. This bar does not add enough transitional changes for an aesthetically smooth transition. Although the transition is smooth musically, it feels overly cautious."

From the musical context, it is clear that "aesthetically smooth transition", refers to (1), while the second use of the term, "smooth musically", refers to (2). The assumption appears to be that (2) is good and both (2) and (1) is better.

Occasionally, an insightful comment was made regarding the nature of the two forms of smoothness, for example, participant one on the second example:

“It seemed like a more human composed transition because each section was internally coherent (in the main) and morphing “attention” was directed to different tracks at different times, rather than to all tracks at the same pace.”

This explains an important compositional mechanism behind the two different approaches to smoothness.

In the majority, (2) was necessary for the morphs to be judged as effective, while (1) was enjoyed more as added interest, however, some participants preferred (1) despite absence of (2). The quantitative evaluations of smoothness cannot be used, due to the statistically insignificant sample size and the above terminological confusion. Tables in Appendix E (Figures E-3 and E-4) condense all judgements of smoothness and effectiveness. In these tables, the original definition (1) is default and where the respondent appears to have meant meta-smoothness (2), it has been annotated as such.

Perceived changes and responses

Participants were asked for subjective responses to important changes, with musicological justifications. I classify the changes as either “variation” or “layering”, where variation involves note adjustments and layering involves the introduction or removal of parts. There were a variety of reactions, with most preferring human composed layering techniques rather than *LEMorpheus* variations. Despite this, particular instances were popular and there was a great mix of opinion. Condensed results are available from Appendix E (Figures E-5, E-6, E-7 and E-8).

In the ***LEMorpheus* morph for the first example**, the most notable variations were in the rhythmic changes from four to seventeen seconds. For some, the variation was interesting foreshadowing or – in combination with phrasing, texture and timbre – acted as a bridge. For others, it was unexpected, abrupt, or the result of incompatible beats. One view was that it was too predictable; however, it is unclear if this referred to the key changes or the rhythm. Most participants had positive reactions to variations in tonal parts, in particular, the bass.

In terms of layering, the entrance of new material induced a negative initial reaction in most people that became more positive over time. This included the cowbell sound early on at three seconds, dissonance between bass and pads at six second and others. Occasionally, entrances were appreciated, in particular, the bass note that enters at seven seconds. Shifts in focus were also observed, for example, from sound effects to the accompaniment at eleven seconds.

In the **human composed morph for the first example**, 'layering' changes were predominant, for example, the clear cut at fourteen seconds was noted by almost all participants. Most thought it worked well, either because it highlighted the organ, brought the music back to a clear tonal centre or had clear rhythm. Others disliked the organ. However, by twenty-one seconds, many felt the break was too long. Some disagreed, describing the timing as "perfect". In terms of variation, the dissipation of high frequencies early-on at three seconds foreshadowed change for some. Another comment was that the change in bass at eight and ten seconds, without the change in lead, made the tonal centre too ambiguous.

The perceived changes in the other examples, with the exception of example four, highlight similar differences between the *TraSe* and human composed morphs and it is unnecessary to reproduce them here (see Appendix E, Figures E-5 to E-8).

For the **LEMorpheus morph of example four**, a notable variation was the upwards pitch shift to the marimba at six seconds. Some viewed this as a worthwhile and interesting key change, hinting of future variations. Others viewed it as a jumbly, conflicting key change. Another important change was the simplification of marimba and percussion at twenty-three seconds. Most people felt the simplification effectively relieved the metric tension and resolved the target bassline, while one participant stated that the harmony was too simple.

For the middle section, those who focused on the rhythmic section appreciated the fills that were used to punctuate changes, or conversely, felt that more punctuation was needed. Changes in density were also noted, for example, the thinning of rhythmic elements at seventeen was perceived as a logical progression, while the dense drums at thirteen appeared as a signal to change. Comments regarding the marimba melody were that it was an effective bridge, natural, effective combined with the bass.

For the **human composed morph of example four**, an important variation was the marimba at eleven seconds. One participant viewed the pitches to be random and others observed the change to be pleasing and natural, despite being sudden. Another noted the effective synchronisation of rhythm. The marimba and bass at seventeen to eighteen seconds was viewed by some as being clunky or too late. For others it was a pleasing riff that revealed the beat and highlighted the new pattern. The stripping back of rhythm and convergence of marimba and bass at twenty-two seconds was also noted by many, who felt this was a positive change and release of tension. One participant felt that this point was too repetitive and boring. Other observations of changes included the auxiliary percussion fills at seven seconds, which two respondents felt destabilised the rhythmic focus or was jarring, and the shift in focus brought by the off beat synth hocketing with the drums at fifteen to sixteen seconds.

To summarise, there appears to be a diversity of opinion, with positive and negative views on a particular change often being held by an even mix of participants. This often seems due to the fact that different participants are listening to different parts in the music or that they have different levels of tolerance to change. Despite this, differences of composition style between *LEMorpheus* and the human composer were evident: variation was a feature of *LEMorpheus* morphs, while skilful layering was demonstrated by the human composer. However, this distinction did not exist for the fourth example where the source and target timbres were the same.

Real-world applicability of morphs

Almost all participants considered the *LEMorpheus* morphs to be applicable to both computer game and electronic dance music context. The worst result was from participant seven who was unsure in the case of the first two examples and thought the third example would be applicable to computer games but did not think it would be applicable to the dance music. Participants eight and nine showed a greater tendency to be critical, however, they appeared to be more critical of the human composed morphs than the *LEMorpheus* morphs. These results are reproduced in the following table:

Applicability of each morph to computer game / dance music								
Participant	1L	1H	2L	2H	3L	3H	4L	4H
one	y/n	y/y	y/m	y/y	y/y	y/y	y/y	NA
two	y/y	y/y	y/y	y/y	y/y	y/y	y/y	y/y
three	y/y	y/y	y/y	y/y	y/y	y/y	y/y	y/y
four	y/y	y/y	y/y	y/y	y/y	y/y	y/y	y/y
five	y/m	y/y	y/y	y/y	y/y	y/y	y/y	y/y
six	y/n	y/y	y/y	n/n	m/m	y/y	y/y	y/y
seven	m/m	y/y	m/m	m/y	y/n	y/y	y/y	y/y
eight	y/y	n/n	y/y	m/m	y/y	n/n	n/n	y/y
nine	y/y	n/n	y/y	y/y	n/n	y/y	y/y	n/n

Figure 39 Judgements of whether the morphs are applicable to the real-world contexts of computer games or dance music. Responses were originally in natural language, but have been condensed for display within this table. They are in the form: computer game/dance and can be y=yes, n=no or m-maybe.

Modifications to the morphs

Participants were asked to describe how they would improve the examples they had listened to, so as to provide compositional techniques that could be incorporated into future algorithmic developments. While some trends were clear, opinions varied wildly, with some advocating complete overhauls and others suggesting that no improvements should be made – both to the *LEMorpheus* morph and the human composed morph.

With *LEMorpheus* morphs, most participants suggested that, rather than attempting a continuous morph, the changes should progress in discrete segments. As well as this, it was often suggested that the morphs should be “thinned” out.

With the human composed morph, feedback included advice on the timing of changes, the addition of extra layers, removal of specific segments within layers. More often than with the *LEMorpheus* morph, it suggested that no change was needed.

For both the *LEMorpheus* morph and the human composed morph it was occasionally suggested that the morph length should be increased. A detailed collation of these results is in Appendix E (Figure E-9, for the *LEMorpheus* morphs, and E-10, for the human morphs).

7.6.4 Discussion: musical controversy

While some trends have emerged and are discussed above, the most striking feature of the questionnaire was the levels of disagreement, with the most polarised responses to the *LEMorpheus* morph occurring in example two. The complex key modulation and non-standard integration of rhythm from the target seems to have been the cause for most of the controversy. Some relevant quotes are included below:

“Excellent - because of the complexity of the tracks, the highly skilful melding of the source and target was a joy to hear. It was a very deliberate transition in moods, and almost evoked a story in itself as the listener was transported from a relaxed environment with the potential for disaster directly into a schizophrenic sound-scape. Very worthwhile.”

- Participant five, responding to the *LEMorpheus* morph for the second example.

“The key changes is REALLY interesting with the bass. Intense change, uplifting, then dark then resolute into the new key.”

- Participant nine, responding to the LEMorpheus morph for the second example.

“Blechh!! Intense negative. Not effective, an apparently random substitution of elements.”

- Participant four responding to the LEMorpheus morph for the second example.

Conversely, the human composed morph for the same example seems to have been generally regarded as being effective or at least adequate by all participants. Particular criticisms were targeted at specific aspects of the morph, as contrasted with the less specific negative responses to the LEMorpheus morph:

“o.k. but not brilliant, the source and target loops were out of sequence with each other - i.e. if you imagine each loop as having groups of four bars, logically bar one of the source would overlap with bar one of the target - this didn't happen, throwing my sense of keeping in place, and making me notice the changeover in a negative manner.... It felt like a mistake that the creator failed to recognise and so didn't fix.”

- Participant three, human transition for the second example.

One explanation for such differences in opinion is that participants were listening to different layers and elements and thus based their opinions on specific features, which are more or less pleasing in general. This is supported by the data gathered from the ‘perceived changes and responses’ section of the questionnaire, where, in certain cases, it becomes clear that the listener is tuning into a particular part, for example, the bass. Another explanation is the varied stylistic preferences of the participants.

7.6.5 Conclusion of formal qualitative evaluation

The formal qualitative evaluation investigated morphing between electronic music loops of various styles. The method to create the music was to manually compose four source and target examples, generate morphs between them using *LEMorpheus*, and hire a composer/producer to

create benchmark morphs. The questionnaire asked for: subjective responses regarding the source and target and morph examples; assessments of smoothness and effectiveness of the morphs; subjective responses and analysis to particular changes in the morphs; opinions as to the real-world applicability of the morphs; and possible modifications to the morphs. The background of the participants and additional feedback was recorded at the end.

The results and analysis from the questionnaire fulfilled the objectives:

1. Elements of *LEMorpheus* morphs that were perceived mostly to have a positive impact were mostly the tonal variations, while the note clutter and lack of structure had a negative impact. In contrast, the human composed morph had clear, structured changes which were more widely appreciated.
2. Techniques for morphing were gathered, and are summarised above (Figure 38). Popular approaches were to blend similar elements and remove dissimilar elements and gauge the dissimilarity of source or target before deciding to blend or switch.
3. There was found to be very little correlation between smoothness and effectiveness. For most participants, effectiveness was a priority and smoothness was of secondary consideration, although, for some, the interest generated by smooth morphing appeared to eclipse any deficiencies.
4. There was a near consensus of opinion stating that the morphing techniques would be applicable to games or live electronic music performance.

Future improvements to the web questionnaire

A number of ideas for improvements were generated and they are summarised here. These would make the process more economic, increase the possible scope of the topic and would be fairly easy to implement:

- Use four smaller questionnaires that can be completed independently and count how many people have completed each. For new participants, the questionnaire with least completions is provided. This would have enabled the sixty or more participants who only partially completed the survey to contribute.
- Use a preliminary focus group to ensure the source and target material is mostly well-liked and credible. In the questionnaire, if they happen not to like the source and target material, ask them to put themselves in the position of someone who does like

it, so as to ensure some kind of useful analytical data is still generated.

- The example with source and target of the same timbres should be shifted to be prior to the other examples with different timbres, to minimise the possible exaggeration of perceived similarity for this example.
- Include a continuous online *forum*, focusing on the changes section of the questionnaire.
- Incorporation of qualitative research software such as *NVIVO* (Richards 2007) to analyse large amounts of qualitative data through semantic association networks.

Ideas for related questionnaires

The results also beg a number of questions which could be addressed in various divergent questionnaires:

- Develop a questionnaire that exclusively examines the creation of hybrid music, rather than hybrid transitions.
- Morphs with the same timbre only. As part of this, ask the participants to rate how different the source and target music is, in order to find the limits of our ability to perceive dissimilarity when restricted to the same timbre.
- It would be sensible in subsequent studies to include a more even balance of female and male responses, rather than a single female.
- Investigation into what kinds of sequences and sounds can work as structural cues, so that an algorithm could be used to generate new cues according to the fundamental principals of what is a good cue.
- A study focused exclusively on rhythmic integration techniques, due to this important element currently being relatively deficient within *LEMorpheus*.

7.7 Extensions

The ‘holy-grail’ for *TraSe* is to define a comprehensive set of transformations that are able to turn any particular source into any particular target in any specified number of steps, without incorporating data explicitly from the target. While this is a difficult and possibly unreachable long

term goal, some clear possibilities for improvement have become apparent nonetheless.

Possible extensions to the *TraSe* morph include: note thinning, note clustering, changes to the transformation chain, structural morphing, automation of parameter settings and optimisation of the *TraSe* algorithm. Note thinning is required to reduce clashes. Clustering notes into phrases would assist morphing of long and dense note sequences. New transformations and strategies for the ordering of transformations could increase the musical possibilities and reduce the number of frames generated. High-level musical structure could improve coherence, particularly in examples that do not afford smooth, continuous morphs. Automatic adjustment of parameter settings would allow the user to find a greater range of usable morphs with less effort. Finally, optimisation techniques could reduce the time complexity of *TraSe* from $O(n^3)$, potentially $O(n \log(n))$, which would enable greater realtime interactivity.

7.7.1 Note thinning

Note thinning would involve the analysis of note sequences before they are played, and removal of notes such that 'clashes' and 'muddiness' is reduced. Such problems contributed to negative responses in the formal evaluation.

Note thinning is not a trivial problem, however, some basic approaches that may alleviate the most noticeable clashes include:

- Removing notes in response to above average note densities.
- Removing or reducing the loudness of notes that overlap or are very close to other notes, even if they are in different parts.
- Constraining the notes within an abstract metre or tonality that is derived from the source and target, weighted on the morph index.

These suggestions may increase the coherence of the morphs to some extent, however, more sophisticated approaches may ultimately be needed, such as automatic mixing. In regards to mixing, human producers have a clear advantage over the morphing algorithm.

7.7.2 Note clustering

Note clustering would involve the automatic segmentation of the note sequence into phrases or 'clusters' of notes. Clusters in the source and target could be paired up and the *TraSe* algorithm would generate a separate list of frames for each phrase pair. This would overcome the

difficulties in applying large-scale transformations, such as *rate*, to note sequences that are long and dense. However, additional care would have to be taken to ensure the phrases are transformed in such a way that they do not clash with other phrases.

Phrase boundaries could be established through temporal proximity, metric position, tonality and the contours for pitch, dynamic and duration; leveraging previous research (Desain, et al. 2005; Lerdahl and Jackendoff 1983). Simultaneous streams of pitches may also be grouped into separate phrases (Bregman 1990).

7.7.3 Transformation chain

A number of transformations have been conceived that may be added to the chain and thus increase the effectiveness and capability of *TraSe*. Some ideas for new transformations that could be developed:

- Differently (perhaps dynamically) shaped contours for *rate*, *phase*, *scale* *pitch* and the others, so that changes occur in different amounts to different parts of the sequence, rather than uniformly across the whole sequence.
- A *rate* transformation that retains the original pitch contour.
- A polyphonic mode for *add/remove* that deals with vertical groupings rather than individual notes.
- Max Reger's techniques for modulation (1903).
- Retrograde (reverse order of notes).
- Inter-splicing of notes.
- Modification of the note onsets using groove quantisation (Chokalis 1999).
- A transformation to change the tonal strength through 'in-scale' and 'out-of-scale' pitch information and harmonic substitution rules.

Additionally, using all possible orderings of the transformations rather than a single linear chain, would increase the range of possible outcomes and results. However, this may impact the speed of the algorithm. If n is the number of transformations, there are $n!$ different possible orderings. With the current eight transformations, this would mean a total of 4,320. This is quite large and certain to have a noticeable impact on the speed of the algorithm, however, it is constant and

thus of much less significance (in terms of time complexity) with respect to the number of notes in source and target, which is variable.

Lastly, the current approach to transform speed works well when there is only one ‘near **perfect**’ transformation (for example, *key/scale morph*), however, there are currently many, ‘far from **perfect**’ transformations in the chain. It should be possible to further augment the calculation of the target dissimilarity to account for the other transformations in the chain, or somehow combine all transformations into one. This may also naturally reduce the extent of dramatic changes in the beginning of the morph.

7.7.4 Structural morphing

As is evident from the formal evaluation (7.6), the current techniques for structuring and layering the music could be improved. This would include components that:

- Determine if the source or target are dissimilar enough to warrant a structural approach, or similar enough for continuous variation.
- Determine the most appropriate points for a layer addition, removal or switch.
- Add cues that foreshadow changes, build expectation for change and reinforce the validity of the change when it occurs.

7.7.5 Automatic adjustment of parameters

An algorithm that automatically adjusts the parameters of the *TraSe* morph to find configurations that produce a minimum of frames is needed. This will increase the musical possibilities of the algorithm, as well as making the process more efficient for the user. The search method would be ‘iterative-deepening’. That is, searching for a parameter configuration that converges in one frame and, if no convergence is found, searching for convergence in two frames and so on.

Another useful automatic adjustment would be to change the behaviour of *add/remove* in response to notes densities. For example, with high note densities, increasing the number of *add/remove* cycles or enabling *add/remove* to operate on larger phrases rather than individual notes would reduce the problem of over generation of frames from dense source and target sequences.

Beyond this, there is potential for a machine learning algorithm to analyse a database of human composed transitions or variations and extract compositional transformations, parameter

configurations and parameter weights.

7.7.6 Optimisation

At $O(n^3)$, the current system is inefficient, however, it could be reduced to $O(n^2)$ and possibly to $O(n \log(n))$. This would enable *TraSe* to compute morphs in realtime, which would enhance the level of interactivity.

The bottleneck is the *add/remove* transformation (7.3.8) and the corresponding Nearest Neighbour (NN) dissimilarity measure (7.3). *Add/remove* is $O(n)$, because it computes the addition of each note in the target and removal of each note in the input. Because each of the n addition and removals is being compared with T using the NN dissimilarity measure, which is $O(n^2)$, we obtain $O(n * n^2) = O(n^3)$.

Reducing add/remove complexity

Add/remove could be reduced to $O(n^2)$, using dynamic programming. A matrix, let it be D , holds the distance measurements between each note of I and T during calculation of NN dissimilarity, $nnd(I, T)$:

fwd	T				
	D	T[1]	T[2]	T[3]	T[4]
I	I[1]	d(1,1)	d(1,2)
	I[2]	d(2,1)
	I[3]
	I[4]
	I[5]	d(5,4)

bwd	T				
	D	T[1]	T[2]	T[3]	T[4]
I	I[1]	d(1,1)	d(1,2)
	I[2]	d(2,1)
	I[3]
	I[4]
	I[5]	d(5,4)

Figure 40 A table for NN distance between I , which has 5 notes, and T , which has 4. The $O(n^2)$ iteration for the forward (left) and backward (right) calculations are shown by the red arrows. Example NNs are highlighted in yellow.

This initial operation is $O(n^2)$, however, by using the matrix D , calculating the change in NN distance when *adding* a note to I from T or *removing* a note from I can be reduced to $O(n)$.

$nnd(I, T)$ (the average NN dissimilarity between the original input and the target) has already been calculated, all we need to do is subtract the distances of NNs that are no longer nearest (for a removal) and add the distances of the new NNs (for an addition).

Reducing the complexity of add

Let x be an index to T of the range $0 < x \leq |T|$ (where $| \cdot |$ indicates the length or cardinality of the sequence) and let add be a function that adds a note to a note sequence. Let D^{+x} be the dissimilarity matrix between the notes in the target, T , and the input sequence with T_x added to it. Therefore, D^{+x} is used for the calculation: $nnd(add(I, T_x), T)$.

fwd	T					bwd	Target				
A_2 (I+ T[2])	D^{+x}	T[1]	T[2]	T[3]	T[4]	A_2 (I+ T[2])	D^{+x}	T[1]	T[2]	T[3]	T[4]
	A_2[1]	d(1,1)	d(1,2)		A_2[1]	d(1,1)	d(1,2)
	A_2[2]	d(2,1)		A_2[2]	d(2,1)
	A_2[3]	...	0		A_2[3]	...	0
	A_2[4]		A_2[4]
	A_2[5]		A_2[5]
	A_2[6]	d(6,4)		A_2[6]	d(6,4)

Figure 41 Finding the NN distance between T and I with a note added from T (the 2^{nd} note in this example) requires one operation for forward (left), and $O(n)$ for backward (right). A_2 is an abbreviation for “ I with the 2^{nd} note from T added”. Grey squares are distances that are subtracted, while squares with a red outline are distances that are added. The number of squares with red arrows going through them indicates the number of operations required.

The **forward** calculation for I with a note from T added, $add(I, T_x)$, is one operation. $av(I, T)$ has already been calculated and the NN will be itself, thus having a distance of 0. However, a renormalisation is required, due to the increased length. We have:

$$av(add(I, T_x), T) = \frac{av(I, T) |I|}{|I| + 1}$$

Equation 6 Forward term simplified for addition of a single note, T_x , from T .

For the **backward** calculation, recall that D is a matrix that holds the dissimilarities between each note in I (rows) and T (columns). Let k be an index to T and the columns of the matrix D , ranging over $0 < k \leq |T|$. For the backward term, we can compare the distance of the NN, $\min(D_{\dots, k})$, with the new distance. If the new distance is larger, the old NN remains; if it is smaller, the new NN is taken. We have:

$$av(T, add(I, T_x)) = \frac{1}{|T|} \sum_{k=1}^{|T|} \min(\min(D_{\dots, k}), dist(T_k, T_x))$$

Equation 7 **Backward term simplified for addition of a single note from T .**

Therefore, the time complexity of the NN dissimilarity measurement for *add* can be reduced from $O(n^2)$ to $O(n)$.

Reducing the complexity of remove

The complexity of calculating *remove* is also reducible to $O(n)$. Let y be an index to the input sequence, I , that ranges from $0 < y \leq |I|$. Let *rem* be a function that removes a note from a sequence and let D^{-y} be the matrix that holds the dissimilarity measurements between *rem*(I, I_y) and T .

fwd	Target				
R ₂ (I- I[2])	D^{-2}	T[1]	T[2]	T[3]	T[4]
	R ₂ [1]	d(1,1)	d(1,2)
	I[2]	d(2,1)
	R ₂ [2]
	R ₂ [3]
	R ₂ [4]	d(5,4)

bwd	Target				
R ₂ (I- I[2])	D^{-2}	T[1]	T[2]	T[3]	T[4]
	R ₂ [1]	d(1,1)	d(1,2)
	I[2]	d(2,1)
	R ₂ [2]
	R ₂ [3]
	R ₂ [4]	d(5,4)

Figure 42 Recalculating the NN distance for I with a note removed (the 2^{nd} note in this example), forward (left) and backward(right). R_2 is an abbreviation for ‘ I with the 2^{nd} note removed’. The row that is crossed out is the row in D that has been removed to make D^{-2} . The number of squares with red arrows through them indicate the number of operations needed. Yellow squares are examples of new NNs, and the grey squares are the previous NNs (continuing the example from Figure 40) that must be subtracted.

The **forward** recalculation involves subtracting the already known NN distance for the note being removed, $\min(D_{y,...})$, and re-normalising by the reduced length.

$$av(rem(I, I_y), T) = \frac{|I| (av(I, T) - \min(D_{y,...}))}{|I| - 1}$$

Equation 8 Forward term simplified for removal of I_y from I .

The **backward** term is difficult to simplify, yet a more efficient implementation has been conceived. Firstly, the NNs for each note in T (the yellow squares in the right of Figure 40) are checked to see if they are were removed (if they were I_y). If so, the distances between the note in T and every other note in $rem(I, I_y)$, are searched in order to find the new NN. Because each note will eventually be removed, all of the NNs will eventually need to be re-estimated and $O(n^2)$ will always occur, but only once in during *add/remove*. This is best described in pseudocode (Figure 43):


```

// Inputs:
T      an array that holds the target sequence of notes
k      the index of the note being removed from the input, I
bwdNN  an array that stores the NN information previously computed for each note in the
        target, T. The index of the note in I that is the NN to the  $i^{\text{th}}$  note in T is
        "bwdNN[i].index". The distance of this NN is "bwdNN[i].distance"
Rm     is I with the  $k^{\text{th}}$  note removed (I without I[k])
// Outputs:
avd    the re-computed average distance of NNs in Rm to the notes in T
NND-BACKWARD-REMOVE(bwdNN, k, Rm, T) returns avd
    FOR(i = 0; i < LENGTH(bwdNN); i++) {
        IF(bwdNN[i].index == k) { // if it was removed
            min // stores the new minimum
            FOR(j = 0; j < LENGTH(Rm); j++) {
                IF(min > DISTANCE(Rm[j], T[i]))
                    min = DISTANCE(Rm[j], T[i])
            }
            avd += min
        } else {
            //if the NN is not being removed, use the old one
            avd += bwdNN[i]
        }
    }
    avd = avd/LENGTH(T) //normalise
}

```

Figure 43 Pseudocode for the *backward* NN distance, optimised for a remove operation.

The ideas above demonstrate how the current complexity of $O(n^3)$ could be reduced to $O(n^2)$, however this is still probably not efficient enough to allow realtime operation. However, the complexity may be reduced further still, to $O(n \log(n))$, through pruning.

Pruning the NN search space to allow realtime operation

Firstly, the notes would be ‘quick-sorted’ according to onset which is known to be $O(n \log(n))$. When searching for the NN of a particular note, notes with the inter-onset distance larger than the Euclidean distance of current nearest neighbour could be pruned:

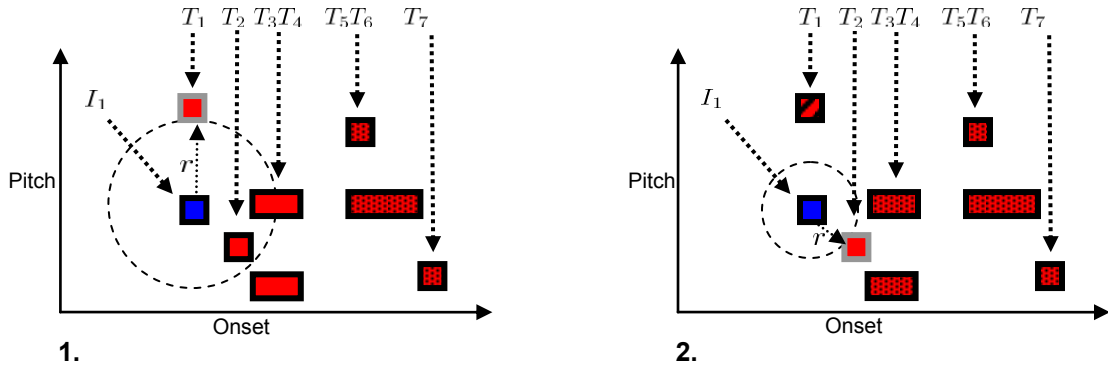


Figure 44 Reducing the search-space when finding the NN of the blue note amongst the red notes. In this example, the blue note is the first note in the input, I , and the red notes are from the target, T . This shows the two steps (labelled 1 and 2) needed to find the NN. The note with a grey outline is the note currently being considered. Notes that are shaded out have been pruned because we know that the distance of their onsets alone will be larger than the Euclidean distance (r) to the current NN (highlighted), because T is sorted. The dashed circle shows the area, within which notes will be closer than the currently considered note.

Figure 44 shows two steps of a search for the NN to the first note of I , amongst T . To start, the distance between T_1 and I_1 (purely vertical in this case) is used to determine the radius r of a circle around I_1 . Any note outside this circle is not the NN. Since T has been sorted, all the notes ahead of T_1 will have a greater start time than it (of course, notes with the same start time would also be considered). All notes with the inter-onset greater than r , can be pruned. This is depicted in step one of Figure 44 where the r for the first note T_1 prunes T_5 , T_6 and T_7 . T_4 (the lowest note), is not pruned, even though we can see it is clearly outside the circle. In step two, T_4 is pruned because the distance of I_1 to the next note, T_2 , produces an r smaller than the inter-onset to T_4 .

The example shown above demonstrates best case scenario. However, if the note we are looking for is at the end of the sequence, rather than the beginning, no significant save on complexity would occur. However, the binary search is $O(\log_2(n))$ and it is likely that this could be combined with the pruning technique.

Other worst case scenarios include extremely vertical music:

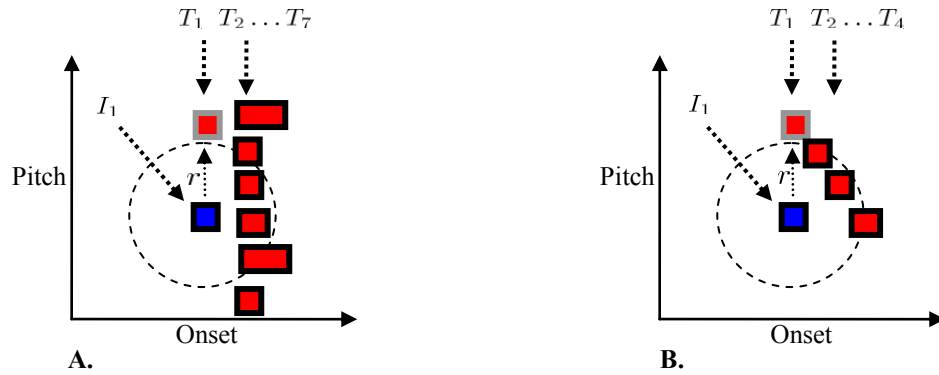


Figure 45 Worst-case scenarios: A. If sorting by start-time, there would be no pruning in ‘vertical’ music. B. If sorting by pitch is also considered, the worst case scenario is a sequence of diagonal notes.

In this case, each note needs to be examined because their start-times are within r . This could be overcome by assessing the ‘verticality’ or ‘horizontality’ of the sequence and sorting according to either pitch or onset, depending on which dimension has the most variation. This technique is promising in that, depending on the music, it could reduce the complexity to $O(n \log(n))$. However, a more detailed examination is needed before any solid efficiency claims can be made.

Finally, it is also plausible that even more significant gains could be obtained through note clustering, however, this technique, may interfere with the ability of the algorithm to converge, due to it being non-optimal. Despite this, exact convergence may not be necessary to musically achieve a suitable morph.

7.8 Summary of evolutionary morphing

To summarise, research into an evolutionary approach to compositional morphing has been presented. This included an explanation of the algorithm, the evaluation of it and possible extensions. The evolutionary morphing algorithm, *TraSe*, was of core relevance to this investigation of automated and interactive compositional morphing of MEM. The *TraSe* algorithm was partially a formalisation of existing theories, for example, it incorporated known compositional transformations such as *rate*, *phase*, *inversions* and harmonies; it used known music representations such as scales, keys and notes; and it modelled the **trial** approach to composition. However, by combining these into a unique morphing algorithm, *TraSe* is also an exploration of new aesthetic possibilities. Thus *TraSe* satisfies the two key elements of the

research objective: formalisation and exploration.

To summarise the *TraSe* process: the source music is put through a chain of compositional transformations repeatedly until it matches the target. The result from each iteration of the chain becomes a **frame**, which is a discrete musical state that could be played during the morph. The series of frames that are generated constitute the morph, progressing from the source and eventually becoming the target. To guide the parameters used in each transformation, dissimilarity measures are used to compare various potential outputs with the target and, depending on user parameters, the candidate which is judged to be most similar to the target is selected as the output for that particular transformation.

Informal evaluation of the *TraSe* algorithm covered musical analysis and automatic testing. The analysis provided some sense of how the various parameters in *TraSe* can be used to influence the musical outcomes. Small differences in the first few frames of the morph can have dramatic effects on the frames towards the end, highlighting the emergent nature of the algorithm. On the level of key/scale morphing, the *TraSe* parameters have a more direct influence on the results and common key modulations can be synthesised by searching for the appropriate parameter combination. The key/scale morphing algorithm can also be used to find unobvious but logical progressions from the key and scale of the source to that of the target.

TraSe was automatically tested using a set of randomly generated source-target patterns, with the goal of converging in the minimum number of frames. The “minimum frames” heuristic is a desirable goal, because when the morph must be of a particular length, a series of frames which exceeds that length is difficult to map coherently into that length. From automatic testing, it was found that while sometimes the *TraSe* morph is more effective than straight-forward addition and removal of notes, it was generally not and would sometimes fail to converge even after one hundred or more frames. Despite this, the test could be improved by using a database of more realistic source and targets, rather than randomly generated ones.

In the formal web-questionnaire evaluation, human composed source and target material was used, human adjustment of *TraSe* parameters allowed and the evaluation involved empirical qualitative feedback from a group of people with musical listening skills. Particular morphs were judged by some participants to be extremely novel and musically innovative. As well as this, the music generated by *TraSe* was considered overall to be applicable to real-world applications such as computer games and electronic dance music. Despite this, the human composed morphs that were created from the same source and target material and used as a benchmark, were regarded as more appealing overall. Participants often differed in opinion because of different foci of attention at particular points in time and differing musical expectations.

Researching evolutionary approaches to compositional morphing of MEM has only just begun and there are many possibilities for future work that would yield substantial gain. This includes note thinning, to avoid muddiness and clashes; note clustering, to allow musical phrasing; new transformations, to extend the range of compositional capabilities; automated layering, to allow higher level structural features such as break-downs; and automatic adjustment of parameters, which would uncover less obvious – but workable – settings and reduce the time spent by the user. As well as these, some ideas for optimisation of the algorithm have been detailed, which could allow truer realtime operation.

Overall, a novel compositional morphing algorithm has been developed that is applicable to electronic dance and computer game music contexts, despite being considered less effective than a human composer. Having explored parametric, probabilistic and evolutionary approaches to compositional morphing with increasing detail and musical success, the research will now be brought to a conclusion.