3 Algorithmic music and morphing

This chapter continues to review the surrounding context, but shifts the focus to explicitly formalised algorithmic music systems. As the goal of the research was to develop new musical algorithms, a review of the field of algorithmic music was necessary so that research opportunities could be defined and influential works highlighted. As with the review of music and morphing in the previous chapter, this acts as a baseline from which the originality and quality of the research can be assessed, as well as providing inspirational ideas. However, in this case, because the objects of investigation are algorithmic, the concerns are technical as well as musical.

Before a detailed comparative study of the various algorithmic music systems can be presented, some terminology and a clear conceptual framework is required. This is provided in section 3.1, which examines previous approaches, both implicit and explicit, before explaining the framework, which was created as part of this research. The new framework operates on two levels – compositional approach and function. The examination of previous frameworks provides some contextual background within which the new framework can be positioned.

Following this, sections 3.2, 3.3, and 3.4 are a series of increasingly narrow reviews where the terms of the framework are employed. Section 3.2 is a broad examination of algorithmic composition, the practice of composing music through a series of formal instructions. While many algorithmic composition projects exist, only the most seminal and pertinent ones are described in detail, while many others are mentioned briefly in passing. This broad review of algorithmic music was relevant to the research insofar as the ideas and techniques for writing musical algorithms could be adapted to more specialised applications such as morphing, as well as providing a baseline from which my own research could be distinguished. As well as this, the range of works cited demonstrate a comprehensive examination of the field and provide a sense of the wider context within which the research operated.

Following this broad review, section 3.3 narrows the scope to interactive music. This is important because of its more specialised relevance to the project. Interactive music, being a younger field, has less established works compared to algorithmic composition but a great deal of recent activity. From the review of interactive music in 3.3 it is clear that many applications could benefit from new techniques for note sequence morphing.

As the novelty and usefulness of note-level morphing is clarified, the question ‘what systems for note-level morphing exist?’ becomes more pertinent. This is answered in the subsequent review

of note-level morphing in section 3.4, providing a historical context which highlights a number of research opportunities.

3.1 A framework for discussing algorithmic music

This section defines a number of terms and concepts that assist in descriptions and comparisons of algorithmic music. The framework presented here essentially combines two perspectives, that of the composer and that of the programmer, discussed in more detail in 3.1.1. The half of the framework that deals with high-level algorithmic music, viewing the music algorithm as a composer-agent, is presented in 3.1.2. Various compositional approaches are discussed. Following this, 3.1.3 presents the other half of the framework which is concerned with low-level algorithmic music. This involves analysing simple algorithms, which may or may not operate within a larger system, in terms of how they relate to musical data.

The framework was an important part of the research in that it enabled the field to be carefully reviewed, which in turn resulted in the discovery of new research opportunities, inspiration for further developments and a baseline from which the novelty of the research could be compared. Before the framework can be fully explained however, it is useful to provide some contextual background about other conceptual frameworks in algorithmic music.

3.1.1 Previous approaches

In this subsection, some of the literature will be reviewed to clarify various existing schools of thought regarding algorithmic music frameworks. A clear conceptual framework does not exist, however there are many publications that discuss algorithmic music to greater or lesser extent and all of them assume some kind of conceptual basis.

Barry Truax (1976) proposed a conceptual framework for music theorists based on careful observations of musical practices. The predisposition of realtime computer music to this form of research was made clear. Truax felt the procedural focus was primarily about relationships between system components (including the human) and could be termed “communicational”.

The *Generative Theory of Tonal Music* (Lerdahl and Jackendoff 1983) examined the psychological basis of hierarchical structures conceived in homophonic tonal music and has significantly influenced the formalisation of music generally. Lerdahl and Jackendoff are careful to present the theory as an aid to music analysis, rather than an *“algorithm that composes a piece of music”* (Lerdahl and Jackendoff 1983, p 5). The connection to the work of Chomsky is in the *“combination of psychological concerns and the formal nature of the theory”* (Lerdahl and Jackendoff 1983, p 5), rather than any particular aspect.

Before any further reference to linguistics, I will briefly explain some relevant terms, initially presented by Chomsky (1956), where the focus was on structural forms. For Chomsky, a **transformation** *"rearranges the elements … to which it applies, and it requires considerable information on the constituent structure"* (Chomsky 1956, p 121). **Generative** was conceived as the property of a finite set of rules that could potentially create, via recursion, a huge or infinite output with a characteristic structure. For example, he examined different forms of rules, seeking to find one that could *"provide simple and revealing grammars that generate all of the sentences of English and only these"* (Chomsky 1956, p.113). In music compositional terms, generative grammars capture the essential logic of particular "top-down" approaches.

In reviewing a number of sound and music programming languages, Loy and Abbott applied programming language and linguistic concepts (1985). The potential of Object Oriented (OO) concepts applied to computer music was identified (Loy and Abbott 1985, p 262-263). Algorithmic music endeavours were distinguished by the level at which the composer influences the code (Loy and Abbott 1985, p 238), for example tinkering with programs, writing libraries or creating languages. This approach highlighted the amount of compositional control but not the different relationships between musicality and procedural forms (which was not the intent of the authors). Pope (1991) also reviewed systems that utilise OO technology, thus focusing on various programming tools and their idiosyncrasies, rather than musical affordances. In contrast, the framework developed for this research addresses the relationship between music and process.

Roads (1985) compiled articles to represent the views of composers working directly with computers. The range of techniques and aesthetic considerations were organised into various topics. Of these, "procedural composition" is the most relevant to this research, where the composer/programmer is less concerned with audible outcomes as they are with the algorithms behind them. However, to Roads at the time, the musically innovative aspects of composing with computers dealt with sound structure rather than traditional concepts such as note organisation (Roads 1985, p xvii). The framework presented here is primarily concerned with note-level algorithmic music due to the potential for innovation within a mainstream musical context.

Desain and Honing (1992) adopted a perspective combining music theory, music psychology and Artificial Intelligence (AI) primarily based on music cognition. The insight that "music is based on, plays on, and makes use of the architecture of our perceptual system" (Desain and Honing 1992, p 6) led this and subsequent studies (Desain, Honing et al. 2005) into the development of generalised musical representations and algorithmic musical processes, especially concerned with analysis. Without being prescriptive, Desain and Honing are careful to limit the scope of their search, necessarily ignoring many aspects of music. However, their view

facilitates deep examination of musical processes by liberating them from idiosyncrasies and enabling comparisons in a general conceptual space, grounded in the architecture of the mind. While also adopting aspects of AI, for example, viewing algorithmic systems as **agents**, the perspective behind this framework is musicological rather than music psychological.

Roads explains algorithmic composition systems and representations in *The Computer Music Tutorial* (Roads 1996, p 821-909). Systems are discussed in terms of history, presentation to the user, interactivity and level of composer responsibility. A special distinction is made between deterministic and stochastic algorithms. Roads also states that, apart from simple cases, there is no perceivable difference between the two (Roads 1996, p 836). Dodge and Jerse (1997, p 341) also examine some "simple cases" and draw the same distinction.

Contrary to this, I argue that some musical mappings are easier to implement than others, for example one-to-one is relatively trivial, which is why particular processes afford, but are not limited to, particular musical outcomes. More importantly, when a system is interactive, normally hidden processes become much more transparent to the audience/user. For example, if a user controls parameters that affect Markov depth or morph index, they will become conscious of their influence and thus more aware of the algorithm underlying the music creation. Thus, for adaptive and interactive systems, addressing the effect of process in music is no longer merely a matter of "compositional philosophy and taste" as Roads stated (Roads 1996, p 836), but an aesthetic and usability imperative.

Laurson introduced the laudable but unreachable aim of musical neutrality for a music software environment in his discussion of *PatchWork* (1996, p 18), as a state where there is no difference between compositional goals and outcomes. I broaden this term to also mean an absence of unintended musical side effects in any implementation of a musical process.

Central to the model of interactive music systems presented by Winkler (Winkler 1998, p 6), are domains of algorithmic music, namely: music analysis, interpretation, composition. Winkler presents his ideas relating to algorithmic composition through the *Max* (Puckette and Zicarelli 1990) visual programming environment, which serves as a specific, idiosyncratic framework for discussing musical process.

Schwanauer and Levitt (1993, p 1-6) conceive algorithmic musical processes as machine models of music and useful tools for scientific examination of theories. Unlike Desain and Honing, emphasis is placed squarely on AI as an increasing source of inspiration for procedural music, with little acknowledgement given to the influence of music theory, practice and psychology. AI, as a computer science perspective, facilitates discussion of efficiency but neglects the musical purposes of algorithms.

In reviewing AI techniques applied to music, Papadopoulos and Wiggins (Papadopoulos and Wiggins 1999, p 1) admit to difficulty in distinguishing works according to the techniques used within them.

> *"The categorisation is not straightforward since many of the AI methods can be considered as equivalent: for example Markov chains are similar to type-3 grammars. Furthermore some of the systems have more than one prominent feature, for example 'Experiments in Musical Intelligence (EMI)' was categorised as a grammar, but it can also be seen as a knowledge-based system or even a system which learns."*

This was the problem that led to the development of the framework presented below, to investigate more appropriate ways of conceiving algorithmic music.

Overall, discussions of algorithmic music are problematic due to the essentially multidisciplinary nature of the field. Algorithmic music sits at the crossroads of musicology, music practice, psychology, artificial intelligence, engineering and potentially other areas. Scholars have not explicitly addressed the problem of a unified framework adequately and so discussions of algorithmic music tend to be idiosyncratic, confused or skewed towards one particular discipline.

3.1.2 Another approach

As shown above, algorithmic music is discussed from a variety of perspectives, often combined, such as linguistic, musicological, sociological, user-centric, artist and programmer-centric (to name a few). The new framework I have developed and presented here is tailored exclusively for the last two, that is, composer/programmers of algorithmic music. Composer/programmers are necessarily occupied both by the musicality of a system and the executably explicit formalisations of musical process. The former can be understood through the compositional approach behind the system and the musical styles exhibited by it, while the latter can be discussed according to how particular algorithms deal with musical data. This is essentially two different levels of encapsulation – Algorithmic Music Systems (AMSs) and Simple Musical Algorithms (SMAs).

An SMA, in the OO sense, is an algorithm at the lowest level of encapsulation, that deals with musical data of some form in either input, output or both, for example, an algorithm that transposes all input pitches by a third (a more detailed definition is given in section 3.1.3). Contrastingly, an AMS exists at the highest level of encapsulation, typically built from a network of SMAs, for example, *EMI* (Cope 2005) or any other major work. Each of these two levels require a different set of descriptive terms. SMAs can be understood according to their direct relationship to musical data, while much larger AMSs deal with different forms of musical data to

such a great extent and variety that it is more profitable to compare them in terms of the overall compositional approach that they model.

The framework for compositional approaches that are built into AMSs is presented first in section 3.1.2, providing a bridge from the discussion of music composition in the previous chapter. The three major compositional approaches that were identified are *abstraction*, *heuristics* and *trial*. Following this, section 3.1.3 explains the terms needed for the more detailed analysis involving SMAs, in particular, the *function* and *scope* of the algorithms.

3.1.3 Compositional approaches of algorithmic music systems

This subsection provides the terms and concepts necessary for a comparative dissection of AMSs in the field and for distinguishing this research from others. Viewing AMSs as "composer-agents" enables meaningful comparisons between these often complex and difficult to categorise entities. While some systems can be differentiated according to their music representations (Wiggins, Miranda et al. 1993), complex works will often use different representations for different parts of the system. As well as this, comparing AMSs in terms of techniques and/or representations tells us little about the nature of musical creativity that exists within the system.

However, if we view AMSs as composer-agents, more musically meaningful questions are afforded such as 'what is its style?' and 'what is its compositional approach?'. These questions are eminently applicable in differentiating systems, as AMSs tend to be designed by people that are inspired by particular theories of creativity and their own personalised approach to composition. While style can be addressed musicologically as per the previous chapter, the notion of compositional approach requires some further explanation. Three different compositional approaches have become apparent – abstraction, heuristics and trial.

Abstraction involves parameterisation and arithmetic, viewing music as parallel continuous parameters upon which arithmetic functions can operate. The parameterisation scheme controls how the music is represented while the arithmetic determines how the data is manipulated. Modelling **heuristic** composition typically utilises a complex network of rules for recombination which are given probabilities extracted from learnt musical experiences. **Trial** involves many iterations and a similar process to evolutionary computing or 'generate-and-test'. During each iteration, some source material may be mutated to generate many possible candidates, from which only a few are accepted.

Even with these terms there is some degree of overlap, but more so when considering algorithmic composition toolkits that encourage multiple compositional perspectives. The toolkits, despite having a broad range of capabilities are less agent-like, being not as focused on higher level decisions as the more specialised algorithmic composition applications which are of more relevance to this research. Frameworks for discussing toolkits are covered by others (Miranda 2001; Ariza 2005).

Overall, the compositional approaches explained below present some clear concepts and terms with which complex AMSs can be compared and contrasted with musically meaningful results, however it should be noted that both composers and composer-agents often combine the various approaches in some way.

Trial: generate and test

Perhaps the most obvious approach to music composition is to engage in an iterative process of generation and selection, trialling patterns and searching for a result that fits a set of criteria. Due to the use of an explicit criteria, it can be viewed as more of a "goal-oriented" form of creativity than the other approaches. This is commonly seen as the basis for many forms of creativity, an attitude epitomised by Thomas Edison's popular quote: "genius is one percent inspiration and ninety-nine percent perspiration". As noted by Gartland-Jones (2002) the most prominent figure from musical history characterised by this approach is perhaps Beethoven, "who left evidence of his constant exploration and reworking in his many note books" (2002: p 14.2). In computer music composition, Hiller and Isaacson (1958) were probably the first to incorporate this approach in the "generate and test" component of *MUSICOMP*.

This approach is particularly well modelled by algorithms that utilise evolutionary techniques. That is, the musical data may be transformed and/or combined (as is the case with genetic algorithms) in a variety of ways, producing a pool of possibilities that may or may not be selected. Each of the candidates are rated according to some kind of **fitness function** and only those that fulfil the criteria are chosen for either the next iteration or for the final result. There are a number of examples of this that are discussed in the review later, most notably, GenJam (Biles 2004).

Within the research I conducted, the final and most successful morphing algorithm was modelled on the 'trial' approach to composition – the *Transform-Select (TraSe)* algorithm, which is detailed in chapter seven.

Heuristic: estimation and rules

Despite the straightforward simplicity of trial and error, some of the most notable composers appear to operate through intuitive heuristics rather than exhaustive variation. The heuristic approach emphasises fast, implicit thoughts and actions that *fulfil the intention* of the composer – a subtle difference to *achieving the goal*, which is the more explicit attitude of the trial approach (explained above). The historical composer most emblematic of composition through a heuristic approach is Mozart, who was able to conceive major works on the instant (Gartland-Jones 2002, p 14.2). Interestingly, one of the early examples of this approach being formalised in algorithmic music is Mozart's Musikalisches Würfelspiel in 1787 (Roads 1996), while the most well documented and possibly earliest example is Kirnberger's 'Method for Tossing Off Sonatas' (Newman 1961).

While both Gartland-Jones (2002) and Jacob (1996) view this form of creativity as being difficult or almost impossible to comprehend, I contend, along with David Cope (2001) that composer-agents that operate through recombination and complex sets of rules learnt from numerous observations are already on this path. A clear example is *EMI* (Cope 1996), which essentially extracts note-by-note probabilities from a huge database, enabling it to compose new works in the style of existing ones. While experts can sometimes detect stylistic inconsistencies, I have found the examples to be surprisingly convincing to most people. While there have been a great number of informal demonstrations, there still has not been any formal "Turing test" of Cope's work. It is clear that Mozart had an exceptional memory for music, being able to replicate entire works from only one hearing, and therefore is likely to have had some kind of internal "database" of his own (Slodoba 1985). Composing from probability involves "note-by-note prediction", along with complex rules that can take into account the musical context on many different levels as it unfolds, so as to ensure that the result has some form of coherence. While the database analysis may take some time, the generation is virtually instantaneous, with one operation per musical unit (note or beat). As a result, there is no evolutionary component and no "perspiration" – only implicit heuristics. This model of musical creativity is further supported by the evidence that enjoyment of musical listening results primarily from the interplay between prediction and perception, as purported by computational musical psychologists such as Huron (2006). Examples of algorithmic music that could be said to model the heuristic style of composition, including *EMI*, are discussed within the reviews (3.2, 3.3 and 3.4) below.

The second algorithm that was developed as part of this research, the *Markov Morph*, was based on the heuristic approach and is detailed in chapter six.

Abstract: parameters and arithmetic

Abstraction in musical composition involves defining relationships, implicit or explicit, between different forms of data, both musical and non-musical. The emphasis is more on exploration rather than goals or intentions. The approach has been employed occasionally by composers throughout history, however it became much more widespread since the adoption of the computer as a compositional tool in the latter half of the 20th century.

The mapping can work both ways, for example, one might map syllables to pitches, an approach used by Guido d'Azzero around 1026 (Kirchmeyer 1968); or one might take musical elements and manipulate them as numbers, as was practiced by the "total serialists" such as Boulez and Webern. For example, pitch, dynamics, duration and onset were all able to be serialised and manipulated through arithmetic. The computerisation of music since the 1950s has enabled a virtual explosion in this approach to composition, for example, the adoption of MIDI in 1983 provided a widespread standard for the parameterisation of note-based music. When dealing with MIDI, one cannot help but be concerned with issues of mapping, whether on the level of sound-to-gesture or, less commonly, algorithm-to-music.

"Top-down" composition (Miranda 2001) is another form of mapping. With this approach, the composer plots the intended levels of intensity through the duration of the piece and demarcates contrasting themes – high-level parameters. From these parameters will be crafted sections of music that correlate more or less to the intensity, according to the composer's judgement. Music has been conceived in this hierarchical fashion by many composers, for example, Burke, Polanky and Rosenboom's Hierarchical Music Specification Language (HMSL) (2005) allowed for two dimensional Cartesian "shapes" to be applied to different levels of the musical hierarchy. Other approaches that involve mapping as part of the compositional approach are discussed in the reviews in sections 3.2, 3.3 and 3.4.

Within this research, the algorithm most related to the compositional approach of mapping is the "parametric morphing algorithm", which is explained in chapter five.

Summary of compositional approaches of AMSs

AMSs, when viewed as composer-agents, can be compared and contrasted according to the compositional approach that they model. I discern three different approaches to musical creativity – trial, heuristic and abstract. Trialling involves transformation and filtration of the music; heuristics involves estimation, rules, recombination and probabilities; abstraction involves arithmetic and parameterisation.

It might be argued that this framework is unnecessary – 'trial' could just as well be understood as 'evolutionary'; 'heuristic' as 'stochastic' and 'abstract' as 'deterministic'. However, these labels position the musical algorithms as computational, rather than musically creative processes. As well as this, they are partially inaccurate – abstraction can involve parameterisation of probabilistic data, while heuristic approaches often include deterministic rules, particularly in clearly defined musical styles such as polyphonic harmony, as explored by Hiller and Isaacson (1958) and Koenig (in Roads 1996). Nor does the trial and error process necessarily involve biological evolutionary concepts such as reproduction, mutation and recombination, particularly at the most primal level of "generate and test".

All of the AMSs that are reviewed in sections 3.2 to 3.4 can be described and differentiated according to these three approaches. However, while the terms are useful for descriptions of AMSs at a high level, discussion of the low-level SMA components requires some more detailed terminology which is provided below.

3.1.4 Attributes of simple musical algorithms

The previous section presented some terms for discussion of compositional approach, however, it did little to aid discussion of low-level music algorithms. To fulfil this need, suitable descriptive terminology for Simple Music Algorithms (SMAs) will now be explained. In particular, two continuums are introduced: **function** and **contextual breadth**. The function continuum describes what the algorithm does to the music and ranges from **analytic**, through **transformation**al to **generative**. The SMAs are defined as algorithms that occupy a single point on this continuum. This is in contrast to the more complex AMSs that are able to occupy the whole range of the function continuum and were the topic of the previous section. The contextual breadth continuum relates to how much contextual information is at the disposal of the algorithm, ranging from narrow to broad.

In order to provide a tight definition of the functional continuum, it is first necessary to examine the various ways music is represented and, in particular, how the representations can be changed to be more or less predisposed to the conveyance of "musical" information.

Musical predisposition of representations

At a conceptual level, musical predisposition is a measure of how well a musical representation can portray the musical ideas and facilitate particular compositional processes that are required. It is an important concept, as the position of an SMA on the functional continuum is defined by the differences in musical predisposition of the input and output. If one were so inclined,

predisposition could be derived formally by ascertaining the lack of complexity in applying the representation to specific compositional processes. Gauging algorithmic complexity itself is a well defined area of computer science (Russell and Norvig 2004) which need not be reiterated here.

A representation may become more predisposed to a certain musical process by adding relevant parameters or structuring the representation more appropriately. For example, in a note representation that only consists of duration and pitch, adding a vibrato parameter would increase the predisposition towards musical processes that incorporate expression. Representational structure is important also; consider the difference between a sequence (ordered) and a set (unordered) of note events. Without some additional computation, it is impossible for the set to represent the musical structure of a sequence containing the same events. Therefore, sets of individual notes have less predisposition to musical tasks such as "create an eight bar phrase" than do sequences which may already be part way organised as a phrase. To reiterate this claim on a macroscopic level, a large database (set) of licks is not yet a piece of music but a sequence of licks can be. Many other sonic, performative and structural elements affect the musical predisposition of representations and have implications for the nature and potential of the algorithms using them.

It is acknowledged that musical predisposition, if implemented as an absolute measure, fails to represent the specialised stylistic aspects of disparate musical implementations. Instead, this framework only ever uses predisposition as a relational indicator. That is, a representation can only have "more or less" musical predisposition. A systematic method for obtaining a more general musical predisposition of a representation could be to define the musical predisposition of a large group of AMSs and compare the single representation to the average; however, for the purposes of this research, such definitions are not needed.

Analytic Algorithms: reduction

Analytic algorithms tend to reduce the potential data size and the general musical predisposition of the representation by extracting specific features. For example, taking a set of sequences and outputting a set of notes can be considered analytic. The set of notes could be a scale that has been distilled from a database of riffs. Another example could be Schenkerian analytical processes such as deriving a single 'Urlinie' or 'fundamental bass' line (Lerdahl and Jackendoff 1983; Cope 1996), or even just a chord (Thomson 1999), from a complete work.

Transformational Algorithms

Transformational algorithms tend not to have a significant impact on the musical predisposition of the data representations but can alter the information. For example, an algorithm that transposes individual notes retains the parameters and structural relations of note collection as representation, but alters the pitch value of each note. A retrograde algorithm that reverses the order of a phrase retains the sequential note representation while transforming the structural pattern, but the music representation of the transformed phrase is unaltered and thus the musical predisposition is unaffected.

Generative Algorithms

Generative in the context of this framework means 'musically generative' – when the resulting data representation has more musical predisposition than the input. For example, a chaos music algorithm that takes a single number as a seed and returns a sequence of notes that can be played is generative.

Contextual breadth

Context is the surrounding information that influences the computation of an algorithm and therefore an algorithmic music system. In OO programming terms, the context is the world-state data and arguments that the algorithm has access to. Context has a breadth or size which can extend along a continuum from localised processes that are context-free to processes that are highly context-dependent. This definition of context-dependency encompasses Chomsky's (1957), but is applied more generally to continuous as well as discrete symbolic data.

An intuitive way of visualising contextual breadth is to imagine a note positioned on a musical score with a circle drawn around the note that defines the scope of its context. The circle would include previous and future notes in the same part and concurrent notes in adjacent parts. The size of the circle would influence the breadth of context. In practice, the contextual breadth has more than these two (temporal and textural) dimensions. For example, an algorithm where notes are influenced by four parameters has a broader context than a similar algorithm that considers only one parameter.

Summary of attributes of SMAs

SMAs exist at the lowest level of encapsulation and can be conceived along two different continuums, function and context. Function ranges from analytic, through transformation to

generative; while context ranges from narrow to broad. The terminology that was defined here has been put to use throughout the reviews in sections 3.2 to 3.4 and later chapters. It is particularly applicable to algorithms that occupy a single point on the function continuum, rather than the larger scale AMSs which often span whole sections, however it can also be used when dissecting AMSs and discussing their lower-level constituents.

3.1.5 Summary of the framework

Most of the previous frameworks for conceiving algorithmic music have been expressed only implicitly, or skewed towards a particular discipline such as psychology or artificial intelligence. The framework for this research relates to the two primary areas of interest – music composition approach and algorithmic function, that is, for dealing with both high-level AMSs and low-level SMAs respectively. Three compositional approaches of AMSs that I identified are **trial**, which is characterised by mass generation and selection; **heuristic**, which is based on estimation and rules; and **abstract**, which involves exploration of parameter mappings. Two primary attributes of SMAs are the algorithmic music function, which ranges on a continuum from generative through transformational to analytic; and contextual breadth, which is an indication of the amount of musical data that the algorithm has access to.

This framework was crucial to the research, as conceptualising the set of attributes described above was an important preliminary step towards achieving meaningful comparisons between various algorithms which in turn allowed for a comprehensive review of existing algorithms. This clarified the state of the art and revealed the most opportune research directions, thus enabling the informed development of musical algorithms to enhance adaptivity in the delivery of mainstream electronic music, which was the ultimate intention. The first, most wide-ranging review begins below in Section 3.2.

3.2 Review of algorithmic composition

Having explained preliminary concepts (3.1), this section presents a review of algorithmic composition. As the research was concerned with the development of algorithms that generate music, an examination of techniques that other practitioners used for algorithmically executing compositional decisions was crucial. This allowed trends and niches to be identified which guided the direction of research and inspired developments.

Algorithmic composition can be defined as the creation of music from a formal sequence of instructions. This is a broad category that technically includes all music generation via software. However, the use of the term ‘composition’ rather than ‘music’, implies a contemplative approach

where the process generally occurs in compose-time or through batch-processing rather than ‘realtime’. Realtime systems are reviewed in the following section 3.3 which examines interactive music in particular.

Composition also implies operation on the note-level, through the historical association of western music composition with notation. While not generative on the note-level, DJ agents can be considered as ‘composition algorithms’ on the level of musical form and are included particularly due to the relevance of mixing to morphing.

Some other application contexts for algorithmic composition include composer agents, computer assisted composition tools and sonifications. Because the application determines the music composition goals, using these categories enables an efficient comparison of divergent techniques and compositional approaches that are applied to similar goals.

Composer agents are concerned with mimicking the creative compositional act. Composition tools are designed to extend the capacity and ease with which composers create music. Performance rendering is about breathing life into otherwise mechanical music and focusing particularly on articulation, short term tempo fluctuation and dynamics. Sonifications take any form of abstract information and produce a musical or sonic result that in some way is a sonic reflection of the original source. DJ agents create playlists from music audio databases and aim to mix between the tracks seamlessly.

3.2.1 Composer agents

Composer agents simulate the music composition aspect of human intelligence. Different applications vary in the level of human interaction and the source of musicality. Four works which are significant for various reasons are examined below: *MUSICOMP*, *EMI*, *CONCERT* and *Madplayer*. The first three generate classical music and *Madplayer* deals with electronic music genres relevant to this research. While almost any algorithmic composition could be framed as a composer agent, the examples discussed within this subsection explicitly pursue the goal of human-like composition, while those discussed in other subsections are based on other goals.

MUSICOMP

Likely the first example of algorithmic composition with computers was a system designed by Lejaren Hiller and Leonard Isaacson (Roads 1996, p 830) which produced *The ILLIAC Suite for String Quartet* in 1957. There were four sets of experiments conducted throughout the project, each exploring different aspects of the problem of applying the computer to music (Hiller and

Isaacson 1958). *MUSICOMP* was **generative** and **transformational**, without any **analytical** component. The compositional approach modelled by *MUSICOMP* was partly **heuristic** and partly **trial**. The general procedure in the first two experiments was to modify and constrain "random white-note music" using musical rules that were mostly inspired from counterpoint. Some rules use pre-composed material, for example the insertion of a C chord at the beginning and end, or cadence completion. Other rules were "No parallel unisons, octaves, fifths" or "no more than one successive repeat". The third experiment applied the same generate-modify-select (Alpern 1995) structure to a contemporary musical aesthetic. The fourth experiment used various Markov processes and in some cases a system for intervallic tonality. By controlling the influence of various probability distributions, musical features such as consonance and dissonance could be influenced. The tonality system provided an additional increase in contextual breadth by marking pitches for tonal reference.

David Cope (Experiments in Musical Intelligence)

David Cope has arguably produced the most intensively developed body of research into music composition algorithms through a number of projects, the most well-known being *EMI* (Cope 1991; Cope 1996). In 2003, *EMI* was 'killed' by Cope, apparently to instil the body of work that had been generated up to that point with a greater degree of uniqueness that could potentially enhance its musical value. Other projects include *SARA, Sorcerer, Serendipity, Apprentice* and the most recent, *Emily Howell*. *EMI* replicates historical music styles through sophisticated recombination; *SARA* is a scaled down version of *EMI*; *Sorcerer* analyses music for allusions to other pieces; *Serendipity* creates a database of music by trawling the web for certain forms of MIDI files; *Apprentice* can extract and generate musical form; and *Emily Howell* is an experiment in the creation of new styles, rather than replication of existing styles (Cope 2005). Throughout these projects and publications, Cope has contributed a range of effective techniques to algorithmic composition.

From 1980, *EMI* was used to generate a huge number of classical music scores of differing styles (Cope, Bach, Mozart, Chopin, Brahms, Joplin, Gershwin, Bartók and Prokofiev), many of which can be heard over the internet (Cope 2004). The details of the process behind this music have been explained to some extent already by Cope (1991; 1996; 2005), and so only significant or pertinent aspects will be highlighted here. *EMI* incorporated analytical, transformational and generative components and a broad context. The compositional approach modelled by *EMI* was **heuristic** – the database analysis forms a huge array of rules with probabilities which models the kind of subconscious repository of musical heuristics that composers may use. There was no element of **trial** and error or **abstraction** within the system itself, although, the human selection

of material and parameter/controller tuning no doubt had some additional influence on the compositional approach.

EMI required a large database of music, preferably in the style of a particular composer. Initially, the scores must be hand-treated to clarify ornamentation, time signature, key signature and texture. Schenkerian analysis is applied to extract the fundamental harmonic movement (urlinie) and categorise each beat (division of the bar) into a harmonic function using the SPEAC scheme: statement (s), preparation (p), extension (e), antecedent (a) or consequent (c). Each of the categories is defined by the user with pitches that are expected to be heard within that category at various levels of acuity from background (more fundamental) to foreground. The categorisation is then a process of counting the pitches on each beat that match with the pitches defined by the user for each category, weighting the occurrence of background/fundamental pitches higher (Cope 1996). The dynamic and duration of a note also contributes to the impact it has on SPEAC categorisation for a particular beat. The algorithms used in the SPEAC program itself and possibly within other programs such as *Apprentice*, also use attributes such as timbre, harmony, melody, dynamics, spacing, texture and others in the determination of SPEAC identifiers.

As well as a Schenkerian-type analysis, pattern matching was used to find “earmarks” and signatures – patterns, which are almost gestalt in nature, that recur at different locations throughout the database. The signature pattern is considered as a particular stylistic trait, while the earmark is a structural indicator (for example a cymbal crash). In EMI, a pattern matching process scanned through the database, finding similarities in dimensions, such as pitch, interval, harmony, rhythm and many others that are more complicated and involve removal and generation of notes. The degree of leniency that is allowed in the similarity measurements for each dimension was user-defined.

The music was generated through beat-by-beat prediction, where the probabilities of one group following another are derived through a statistical analysis of the music within the database that has been analysed as above. This is essentially an augmented Markov chain, the primary difference being that additional attributes specifying the relationship of each note to certain important notes that appear at structurally significant points in the future are included, thus imbuing the music with a degree of structural coherence.

While global musical structures were manually incorporated into *EMI*, they are automatically generated in the program *Apprentice* (Cope 2005). *Apprentice* generates structure for a piece through a generative grammar that uses the set of SPEAC identifiers as non-terminal variables and particular rules defined by Cope as the set of productions.

CONCERT

Mozer (1994) published experiments into note-by-note prediction using the *CONCERT* system. The test was to see if a complex connectionist system, without explicit structural representations, could learn structural trends and utilise them in coherent compositions. Note-by-note prediction, like Cope's beat-by-beat prediction, is a **heuristic** compositional process. *CONCERT* utilises **analytic** and **generative**, rather than **transformational**, algorithms.

The system is initially trained to predict the next note in a sequence or training set. To compose music the first few notes of a known sequence are then given and the strongest predictions are fed back recursively into the input, thus predicting an entire composition. The Artificial Neural Network (ANN) consists of an input layer which is provided by the pitch, duration and chordal data, a layer containing a distributed or combined representation of the input; a layer containing pitch, duration and chordal information separately, and a layer containing output probabilities for predicting the next note. In an attempt to provide *CONCERT* with a sense of musical structure, an additional layer contained a distributed reduction of the musical history or 'context'.

The representation scheme was interesting in that it attempted to reflect the psychoacoustic similarities between certain values of parameters. For example, pitch was collectively represented through pitch height, a Circle of Chroma (CC), and a Circle of Fifths (CF). Using non-chromatic representations such as the CF meant that in some cases the distance between, for example, C and G, would be recorded as closer than the distance between C and C#. This concept was extended in the harmonic and rhythmic domains, using the tone series and a "circles of beats" respectively. The circle of beats seemed equivalent to using a swing/shuffle dimension.

Tests with listeners found that, while *CONCERT's* compositions were preferred over that of a third-order Markov chain, they lacked global coherence. Although Mozer concluded by doubting the compositional ability of note-by-note prediction, some suggestions were offered for future research. While work in ANN composition has been carried out since (Chen and Miikkulainen 2001; Eck and Schmidhuber 2002), Mozer's recommendations have not been fully investigated and global structure remains the most significant obstacle to this form of composition (Chen and Miikkulainen 2001). Musical examples that illuminate the problem of global coherence can be accessed online (Eck 2003).

Madplayer

More recently, the *Madplayer* (MadWaves 2004), a hand-held music device with a composition algorithm and synthesiser, demonstrated the use of algorithmic processes to create mediocre

music in a range of electronic music genres. Some aspects of the *Madplayer* have been discussed in the patent application by Georges and Flohr (Georges and Flohr 2002), however this does not provide explicit algorithmic detail. The patent mentions “compositional algorithms applied to musical data” (2002). The compositional approach is likely to be **heuristic** – it appears that tunes are generated instantaneously and from musical material, rather than through intensive **trial** or from **abstract**, non musical data.

The rhythmic patterns of the drum, bass, riff and lead are always two bars or less in length. The harmonic progression can be longer than this depending on the style (two bars long in Drum and Bass and eight bars long in Ambient). The harmonic shifts influence the tonal parts, making the perceived pattern length longer through transposition. Sonic changes such as dynamic filter cut-offs also operate beyond two bars, which also affects the perceived cycle length of the pattern. Generally as the piece or section progresses, layers are added. Some layers seem beyond the control of the user. Fills can occur during transitions between sections or within a section itself depending on the length. There are a few different fill types that are applied during these sections depending on the style: part muting, snare rolls, cymbal crashes and transient musical events such as chaotic lead patterns.

For melodic composition a probabilistic grammar, as opposed to a single probability set, seemed likely. In the “trance” style there were a few different forms the bass line could take, with perhaps two or three different types of stochastic algorithms for bass generation within this style. Clear three over four rhythmic passages within phrases were often present. Random compositional transformations, such as changing the rate of the pattern or quantising it to three over four or two over three, may have been used to achieve this. This is supported by the similar re-assemblage of audio drum patterns. If transformation techniques have been used on drum samples, then it is quite likely that similar techniques would have been applied to MIDI patterns. The first clue that points to this is the fact that although the instrument/timbre used by the melodic part is subject to change, it is not possible to change the instrument for the drums. The drum part must be completely recomposed in order to obtain a change in the drum sound (which of course will change the pattern as well). This indicates that drum samples have been used. If the drums consisted of MIDI patterns then it would be easy and probably desirable to allow the user to change the instrument. Despite this, some sounds are recognisable in other patterns. When the drums are recomposed, various parameters are reshuffled to create a new drum loop that sounds unique: spatialisation (reverb level and panning), delay, pitch, pitch bend contour, volume, volume contour and reversing of samples. Especially in styles such as Drum and Bass, it sometimes seems that different forms of cut and paste type compositional changes have been applied to drum parts. This is due to the strange shifts of dynamic present in some of the drum sounds.

Overall, it is difficult to understand exactly how the *Madplayer* works, however, the techniques are convincing within the bounds of mediocre music composition and it is clearly a substantial development.

3.2.2 Computer Assisted Algorithmic Composition tools and toolkits

Computer Assisted Algorithmic Composition (CAAC) tools are designed to enhance the creative musical abilities of composers and music producers. The existing software has varying degrees of algorithmic assistance, programmability and usability – values which define this form of algorithmic music. A range of CAAC software is available (IBM-Computer-Music-Center 1999; Farbood 2000; Amitani and Hori 2002) and is reviewed in most computer music books (Roads 1996; Dodge and Jerse 1997; Miranda 2001; Rowe 2001).

The first commercial sequencer, aptly titled *Sequencer* (Spiegel 1987), developed in 1985 by Dave Oppenheim of OpCode Systems, provided basic manipulations – add and subtract – over note parameters such as pitch and duration. *M* (Zicarelli, Offenhartz et al. 1986) is one of the earliest examples of a commercial composition tool with a range of extended algorithmic features that is still being developed. Music production applications such as *Cakewalk* and *Logic* are semi-programmable, utilising Cakewalk Application Language (CAL) scripts (Cakewalk 2004) and the Environment page (Emagic 2004) respectively. The *TS-Editor* (Hirata and Aoyagi 2003) is a unique application that partially embodies the Generative Theory of Tonal Music (Lerdahl and Jackendoff 1983).

Open source programs such as *Musical MIDI Accompaniment (MMA)* (Poel 2005) and *Pymprovisator* (now abandoned) (Álvarez 2005) as well as commercial programs such as the original *Band-in-a-Box (BIAB)* (Amitani and Hori 2002; PGMusic 2004) and its more recent competitor *Jammer* (SoundTrek 2005), are programs designed to facilitate the rapid creation of standard music. The user defines the chordal structure of a piece and the various parts that play it. As well as representing a MIDI channel and instrument, the parts contain a large bank of short musical phrases which are combined and harmonically adjusted to match the chord progression. These programs are fundamentally recombinant, like Mozart's *Musikalisches Wurfelspiel* (Roads 1996) although a basic kind of supervised probabilistic grammar is often employed to enhance the process. For example, in *Jammer* the non-terminal parts or musicians contain the non-terminal musical fragments or riffs which are labelled either as grooves or transitions, and which contain the terminal notes. Transitions can be further defined using three sliding scales that indicate whether it is the kind of transition that leads well into riffs, notes, or chords which is

interpreted by the recombinant algorithm. Within grooves, the probability of various riffs occurring can be defined by the composer.

Music Sketcher is a technology preview released from IBM in 1998 (IBM-Computer-Music-Center 1999) and abandoned since. It provides substantial assistance with tonal and harmonic operations. The user inserts and arranges pre-composed blocks of musical data called riffs which are modified parametrically – there is no access to the notes themselves. This could be thought of as a supervised recombinant and transformational system. “Modifications” to standard musical parameters such as velocity, duration, rhythm offset and pitch are applied to the blocks via contour graphs. The different operators (plus, minus, multiply and divide) used to affect these parameters seem to have been chosen for mathematic simplicity rather than musical relevance. In a similar way to *BIAB* and *Jammer*, the harmony track describes the chords and tonality of the sequence. The *Smart Harmony* system (Abrams, Oppenheim et al. 1999) performs an analysis on the original riff, and re-maps the pitches to suit the tonality and chord progression (IBM-Computer-Music-Center 1999). Many musical examples have been created with *Music Sketcher* and can be viewed within the program itself, which is available on the CD accompanying the book *Composing Music with Computers* (Miranda 2001).

Lower-level tools and libraries such as *Nyquist* (Dannenberg 2002), *CSound* (Vercoe 2000), *Common Music* (Taube 2004), *AC Toolbox* (Berg 1992) and *jMusic* (Sorenson and Brown 2000; Sorenson and Brown 2004) are the most programmable and algorithmically sophisticated, although, they typically lack usability in that they require the composer to have knowledge of programming. Extensions to lower-level tools such as *CSound*’s *Cecilia* (Burton, Piché et al. 2004), my own sequencer for *jMusic* in *LEMu (Live Electronic Music)* (Wooller 2004) and the shape editors in *AC Toolbox* (Berg 1992) are attempts to overcome this problem by incorporating a Graphical User Interface (GUI).

Programming languages where the GUI reflects the data flow are called Visual Programming Languages (VPLs). VPLs related to music and sound include: *Algorithmic Composer* (Fraietta 2001), *Max/MSP* (Cycling'74 2004), *Pure Data (PD)* (Puckette 2005), *jMax* (Dechelle, Schwarz et al. 2004), *Keykit* (Thompson 2007), *Autogam* (Bachmann 2001), *Patchwork* (Laurson 1996), *OpenMusic* (Truchet, Assayag et al. 2001), *Plogue Bidule* (Plogue 2004), *Reaktor* (Native-Instruments 2007) and *Karma* (Klippel 2005). Both *Max/MSP* and *PD* are well supported and contain a constantly growing list of objects (Chamagne and Ninh 2005; Puckette 2005). These programs become particularly powerful when using externals which are objects programmed in *C* according to a specification that makes it usable from within the VPL. Some objects in Max that are relevant to the techniques I have used in this research are the implementations of Markov Matrices: *Algo1, markov, markov-harmony, markov-rhythm, markov~*, *mchain* as well as

the genetic algorithm toolkit, *gak*. Objects related to morphing are either related interpolation of presets, for example, *l.preset* and *plugmorph* or audio, for example, *lp.tim~, lp.vim* and *morphine* (Chamagne and Ninh 2005). *OpenMusic* has been developed primarily as a visual tool for experimenting with musical constraints. *Bidule* includes basic stochastic note generation capabilities and the more complex "particle arpeggiator" that models physical particle properties to create a stream of MIDI notes. It is effectively a chaos note generator with controls over speed, density and pitch tendencies. Many of the other systems mentioned include similar algorithmic processes.

CAAC tools that deal specifically with the application of adaptive music such as *DirectMusic Producer* are dealt with in the section on adaptive music and audio (3.3.2).

3.2.3 Sonifications

Sonifications are attempts to derive music or auditory data for scientifically motivated listening from non-musical processes such as mathematical formulas or brain wave signals. The musical success of a sonification depends on the level of compositional involvement, the design of the mapping from raw data to musical data and, to a lesser extent, the design of the abstract processes themselves. Most sonification algorithms are an **abstract** rather than **heuristic** approach, but some works incorporate evolutionary procedures and thus could be seen as **trial** based, in the sense of the compositional approaches described above (3.1.2)

Pioneering work in sonification was conducted by Iannis Xenakis from the 1960s (1971) who applied stochastic processes to waveform and note data. A sense of musicality was expressed through compositional arrangements and the parametric control of constraints such as frequency range and amplitude. Within the development environments of the VPLs mentioned above, a plethora of idiosyncratic mathematical sonifications have been created. More specialised applications for sonification such as *Tangent*, *Texture*, *MusiNum*, *A Music Generator*, *FractMus*, and *Vox Populi* have been reviewed by Miranda (2001), including his own *CAMUS (Cellular Automata MUSic)*. These and others such as *LMUSe* (Sharp 2006) and *MetaSynth* (UI-Software 2005) utilise processes ranging from number theory, fractals, cellular automata and genetic algorithms. Other systems sonify and visualise complex evolutionary processes. For example *Gakki-mon Planet* (Berry, Rungarityotin et al. 2001) generates music from non-musical world state parameters. Those systems that apply evolutionary procedures such as *Vox Populi* and *Gakki-mon Planet* could also be viewed as embodying the **trial** approach to composition, in addition to the **abstract** approach.

3.2.4 DJ agents

DJ agents aim to mimic some or all of the musical tasks utilised by DJs, including track selection and sequencing, beat matching, cross-fading and mixing. Automatic DJ algorithms can be seen as part of "algorithmic composition" insofar as DJing is recognised as a compositional activity, which, without diverging overly, I assume is at least partially the case.

Cliff (Cliff 2000) created a dance music DJ agent which, given a small set of indicative tracks or some qualitative suggestions, could select and sequence an entire DJ set from a database and seamlessly mix between them. What is described by Cliff appears to be a sophisticated DJ agent, although demonstration software could not be found. Through the connectionist analytical techniques, this system predominantly models the **heuristic** compositional approach. To a lesser extent, the mapping of qualitative data could be considered part of an **abstract** approach. Fujio and Shiizuka's (2003) DJ agent employed an interactive evolutionary computing approach in an attempt to adjust the cross-over point to suit the personal aesthetic human sensibilities of the user, which is a **trial** approach.

The other systems that were examined (Fraser 2002; Andersen 2005; Jehan 2005) tended to focus on technical aspects of DJing and were thus unable to be adequately discussed in terms of compositional approach. As low-level music algorithms, they are mostly **transformational** and **analytic**, rather than **generative**. Jehan (2005), as part of the *Skeleton* software, developed an automatic DJ with tempo and downbeat extraction, alignment, pitch-independent time stretching and cross-fading. An algorithm was employed to find rhythmically similar sections within the two pieces that would be the most suitable for a transition – a fairly **broad** context that is available to the algorithm. Jehan introduced the option of mid-mix tempo adjustment, a technique which results in an aesthetic which is fairly alien to electronic dance music. This is due to the limitations of traditional turntable technology, as I discovered through the qualitative "focus concert" (see chapter six). Fraser (2002) created an automatic DJ, with algorithms for dealing with tempo extraction, beat alignment and cross-fading. The cross-fade volume envelopes used during the transition could be drawn by the user, in contrast to typical DJ cross-fades which are limited to "equal-energy", that is, logarithmic or inverse-exponential curves.

3.2.5 Summary of algorithmic composition review

An overview of algorithmic composition has been presented, highlighting a number of application contexts and broadly examining the field. Most of the significant composer agents analysed in 3.2.0 appeared to model the **heuristic** approach to music composition, implying a niche for

composer agents that model other compositional approaches. The works all involved **generative**, **transformational** and, to a lesser extent, **analytic** algorithms.

The range of CAAC tools and toolkits that were examined in 3.2.1 highlighted the level of diverse activity within the area and the kinds of basic algorithmic techniques that are used. The compositional approaches and music algorithm attributes mentioned in the framework (3.1) were evident, however, the level of sophistication was fairly low, due to the focus on enabling the user to create their own, more complex and specialised algorithms. It is clear that CAAC, as a diverse amalgamation of tools for composition and musical experimentation, can be a niche for any kind of algorithmic music technique, including note-level morphing.

The sonification systems that were examined in 3.2.2 mostly embodied the **abstract** approach to composition; however, some could also be seen partially as **trial**, based on evolutionary processes in the world-state being abstracted. Most sonifications tended to utilise **generative** and, to a lesser extent, **transformational** algorithms. The musicality of the sonifications I studied were mostly related to the mappings that were chosen. Sonifications are experimental in nature, and therefore have no economic need for specific techniques; nonetheless, novel techniques add to the experimental capabilities of sonifications. Note-level morphing could be applied to sonification as a method of combining or progressing through the musical results of different mappings.

Most of the DJ agents studied in 3.2.3 modelled only the low-level technical skills of DJs, with predominantly transformational and in some cases analytic algorithms. In the cases of higher-level, more artistic skills such as track ordering and determination of cross-over point, some works utilised the **heuristic** or **trial** approach to composition. Note-level morphing in DJ agent research is not apparent – despite the obvious correlation between morphing and mixing. This is understandable, as without the source MIDI sequence and synthesisers used to create particular tracks that are being mixed, it is currently very difficult to apply note-level techniques to the waveform-level practice of mixing. Nonetheless, the examination of DJ agents provided some insights into mixing techniques that might be applied to morphing, for example, automatic determination of the cross-over point and logarithmic crossfades.

In summary, a review of the field of algorithmic composition has been presented, comparing a number of algorithmic composition works in various application contexts. The process of reviewing the field has enabled particular trends and research opportunities within each context to become visible. This information was used throughout the research for inspiration and to gauge the novelty of the techniques that were subsequently developed.

However, algorithmic composition is a broad field and a narrower review that covers the newer field of interactive music in more detail is also necessary due to its particular relevance to the interactive nature of the research.

3.3 Review of interactive music

Having covered algorithmic composition, the review focus will now shift to interactive musical algorithms. Compared to the review of the previous section, it comprises a collection of relatively newer works with narrower application focus. This is particularly pertinent to the interactive nature of the project. Discussion of interactive musical algorithms that are specifically related to note sequence morphing is reserved for the review of note-level morphing in the following section (3.4).

Interactive music contexts involve transfer of information between some external source and the algorithm which can influence how the music is generated while it is being heard. Rowe (1993) developed a taxonomy of Interactive Music Systems, involving three attributes: score driven vs performance driven (level of sponteneity); transformative, generative or sequenced (compositional techniques); instrument vs player (the level of autonomy).

While Rowe's taxonomy is useful for description and analysis of interactive music systems in general, I have instead arranged this review in terms of contexts to which note-level morphing could potentially be applied. Such application contexts include users guiding meta-instruments (3.3.0); musicians jamming with jamming agents (3.3.1); computer game players influencing the adaptive music of games (3.3.2); or participants experimenting with interactive installations (3.3.3).

A meta-instrument is similar to a musical instrument in that it is controlled directly by a musician with the intention of producing a particular musical effect. However the meta-instrument provides high-level parameters that are used to guide the overall music rather than requiring the musician to specify each individual note. Jamming agents take on the role of an improviser – listening to the music of the other musicians and formulating a response in realtime. Adaptive music systems or 'stage pit agents' observe a scene that is unfolding and formulate an appropriate response in realtime. The term adaptive music or adaptive audio is used in computer game music literature. Interactive installations are artworks involving physical interaction with visual and/or sonic processes.

By reviewing each context, numerous opportunities for applying note-level morphing became evident and some approaches to interaction that were found in the various works were inspirational to developments within this project.

3.3.1 Meta-instruments

Meta-instruments are designed for the live delivery of algorithmic music, taking instructional input from a player and rendering it into musical output. For systems that achieve this through a direct, **narrow** context mapping of gesture to sound, interactive function is more similar to traditional instruments. Meta-instruments are generally more sophisticated, **broad** context systems that take advantage of the algorithmic possibilities of software. In situations where the instruments, "meta" or otherwise, are combined with a suitable physical interface, the system is cybernetic (Pressing 1992).

The *KARMA* (Kay 2004) keyboard includes generative and transformational algorithms that the user tweaks in realtime to conduct a performance. *Jam2Jam* (Brown, Sorenson et al. 2004) is a meta-instrument based on stochastic generative techniques. Users control compositional parameters such as note density which affect weightings within each part. A basic system of harmony has also been implemented. In generating notes rapidly from musical rules and probabilities, both of these reflect the **heuristic** approach to composition.

LEMu (Wooller 2004) uses transformational and generative algorithms for the live compositional manipulation of electronic dance music. It is based primarily on deterministic transformational algorithms, but also incorporates a probabilistic grammar for generation of breakbeat style rhythmic patterns (Wooller 2003), a **heuristic** approach to composition. The influence of the transformational algorithms on the music is directly controlled by the user. Because of this, whichever approach the user adopts is reflected, be it **heuristic**, **abstract** or **trial**. A similar situation exists for live music production environments such as *Live!* (Ableton 2004), which are also less "meta", having more direct forms of interactivity.

The *Metasurface* feature of *Audiomulch* enables any number of parameter snap-shots to be taken and positioned on a two-dimensional plane. The user can then morph between them through natural neighbour interpolation (Bencina 2005). Although the parameters relate to audio manipulation rather than music composition, some aspects are borderline, for example a Low Frequency Oscillator (LFO) mapped to amplitude will effectively control the rate of occurrence of a sound event in a rhythmic way. Despite being on the audio-level, the *Audiomulch* example is worth mentioning as it refects an **abstract** compositional approach – in the mapping of Cartesian co-ordinates to parameter snap-shot interpolations, the focus is less on matching particular musical criteria (as per the **trial** approach) or being guided by **heuristics**, but on exploring the emergent musical properties of the mapped surface.

Overall, the occurrence of note-level morphing in meta-instruments appears to be rare and those meta instruments that do (Momeni and Wessel 2003) are discussed in the review of note-level

morphing below in section 3.4. The techniques of note-level morphing support meta-level control of the music through the morph index and could therefore be easily applied to a meta-instrument. This is demonstrated by the live concert performance of *LEMorpheus*, the software I developed through this research, as discussed further in the conclusion.

3.3.2 Jamming agents

Jamming agents fulfil the role of a music 'player' (Rowe 1993) involved in an improvisational situation. They take musical input, analyse it, and respond with musical output, necessitating both **analytic** and **generative** algorithms. **Transformational** algorithms are not strictly necessary, but often used.

George Lewis (Lewis 1999) developed jamming agent software from 1985 to 1987 to respond to his solo trombone playing in the CD *Voyager* (Lewis 2007) and has continued developing the software to present day. Robert Rowe, in the seminal *Interactive Music Systems* (1993) and later work (Rowe 2001) discussed techniques and approaches related to his *Cypher* software.

Winkler (1998) conducted similar explorations using his *Followplay* software. The *HybriD* patch (Adam 2002) is designed to respond and resynthesise the performance (MIDI) and acoustic (audio) data gathered from a large ensemble. The *fiddle* object (Puckette and Apel 1998) in *Pure Data* is used to track pitch in jamming agents. None of these jamming agents have been submitted to any kind of testing and, from listening to the music generated (Lewis 2007; Rowe 2007; Winkler 2007), they appear likely to fail "Turing test" style conditions[1]. This is not unexpected, as these works are intentionally experimental, being conducted in the exploratory spirit of the **abstract** compositional approach.

Score-following is the analysis of live input to gauge the appropriate tempo at which to play a backing score and has received significant attention over the years (Dannenburg 1984; Vercoe 1984; Raphael 2003). While tracking techniques are relevant, the lack of generative algorithms excludes score-following from classification as a true jamming agent, being more of an 'accompanist agent'.

[1] The test would determine if the output of the jamming agent is indistinguishable from that of a human musician.

BoB (Band out of the Box) is a jamming agent that learns to emulate the characteristics of the musical style of the human player (Thom 2000). *GenJam* (Biles 2004) is a jamming agent originally based on Interactive Genetic Algorithms (IGAs). *The Continuator* (Pachet 2002) 'continues' half finished patterns that are played and learns musical styles from the musician and/or a database. Due to the high level of musical inventiveness, these last two systems will be analysed in more depth.

The Continuator

Recently developed by François Pachet, *The Continuator* is a virtual jammer that continues musical phrases (Pachet 2002; Pachet 2004). Promising informal tests have been conducted, finding that the human player is indistinguishable from the virtual in most cases. This must partially be due to the fact that the same timbre is used as the human and also that the generated response continues instantly from where the human left off. However, as well as this, global musical coherence is not an important consideration when the system is only required to produce short continuations (Pachet 2005). In this interactive context, the use of note-by-note predictive composition proves quite applicable.

The Continuator learns a Markov model of possible note sequences from a database of music, which can be built in realtime. A key feature is the use of a realtime fitness function to influence probabilities. The weightings of nodes can be adjusted in favour of notes with similar properties to the input stream provided by the interacting musician. The degree to which this occurs is determined by a user controlled variable.

Pachet has approached the problems related to the application of sequential models to polyphonic music by clustering notes within the same temporal region. When generating phrases the user can select from four different algorithms to dictate the rhythmic structure:

- Natural rhythm: negates the clustering, so the rhythm is exactly as it occurred in the database.
- Linear rhythm: streams of eight quantised notes of a fixed duration with no rests.
- Input rhythm: the realtime input is mapped onto the rhythmic structure of the output.
- Fixed metrical structure: clusters notes into beat long segments, regardless of their rhythmic structure

Each of these modes has strengths and weaknesses that are accentuated when applied to different musical styles. Natural rhythm mode can sometimes produce 'unnatural' rhythms when

notes occur outside of their original context. Linear rhythm is useful for certain styles of music where rapid sequences of short notes are common, such as be-bop. It is curious that one should dictate the rhythmic aspects of this style directly, rather than allowing the learning algorithm to extract them from a be-bop database. Input rhythm mode seems especially useful for the realtime jamming situation, providing a greater sense of interaction through rhythmic imitation.

In summary, *The Continuator* uses **analytic**, **transformational** and **generative** techniques. Analysis of musical sequences creates sets of note sequences and probabilities; rhythmic re-mappings transform the music and continuations are generated from probability sets. Because *The Continuator* is influenced by a musical history, musical input and a large number of instructional parameters, the **contextual breadth** is high. The approach to composition is clearly **heuristic** – there is a definite musical intention that is implicit in the Markov model. There is no evidence of explicit, goal-oriented **trial** and error, nor exploration of **abstract** mappings.

The system has been used in live improvisations by Gÿorgy Kurtag Jr., Alan Silva and Bernard Lubat at music festivals such as *Fetive d'Uzeste*, *Sons d'hiver*, *Festwochen* and *Budapest festival* (Pachet 2002). Musical examples can be downloaded from the website (Pachet 2004).

GenJam

GenJam (Genetic Jammer) (Al Biles 2004), is a virtual improviser and accompanist of "straight-up jazz", with the capacity to perform solos, trade fours and provide harmony to a human player. A repository containing structural and motific data for the entire standard jazz repertoire provides the algorithm with a predetermined global structure around which to operate.

Note-level structure is derived from recombination and selection, in the tradition of Genetic Algorithms (GA). Depending on the mode of operation, material for recombination is either derived from a general lick database, a database of licks specific to the current song, or human input. The pitch intervals are applied to a chord progression and scale to determine the real pitch value. The original version was an Interactive Genetic Algorithm (IGA), where the user determined the fitness of the lick combinations. Depending on the mode and level of operation, the newer version of *GenJam* can randomly combine phrases or uses a function to ascertain the most rhythmically appropriate crossover point. A number of other heuristic approaches have been used during combination to ensure a certain musicality in realtime. Biles claims that this avoids the use of a fitness function, however it is not clear to what extent the population pool is reduced and, if it is not reduced to a single member, how the final lick is selected.

An interesting feature of *GenJam* is the set of mutations used in the "trading fours" mode:

> *"GenJam measure mutations include playing the measure backward, playing it upside down, playing it backward and upside down, transposing it up or down a random amount, and sorting the new-note events to create an ascending or descending melodic line. Phrase mutations include playing the measures in reverse order, rotating the measures, and repeating a measure."*
>
> (Biles 2002)

Bile's use of these techniques show how well chosen transformational processes can impart a sense of musicality as well as providing musical flexibility. When I developed the *TraSe* morphing algorithm, which is also based on transformations, the previous success of *GenJam* inspired some confidence that the approach would be feasible. *GenJam* also highlights how data-driven recombination techniques can be used to great effect when the stylistic boundaries are narrowed to a single genre such as "straight-up jazz".

In summary, **transformational** algorithms are central to music creation in *GenJam*, while **generative** and **analytical** processes are important in relation to various musical representations, the generation of lick sequences from lick sets and specific tasks such as determining the cross-over point. The compositional approach is **heuristic** and, to a lesser degree, **trial**, although it is admittedly difficult to assess on the information that is available – the approach seems to have changed through different versions of the software. The augmentation of the fitness function with a set of musical rules clearly brings the approach towards **heuristic**, without trial and error. In the original version of *GenJam*, the use of supervised selection to iteratively filter a population is clearly a **trial** approach.

Biles and *GenJam* gig regularly and some demonstrations – which, in my own opinion, are at least moderately convincing – are available from the website (Biles 2004). In these recordings it is interesting to note that a more straightforward recombinant algorithm is used as accompaniment, in the style of Band In A Box (Gannon 2004).

To summarise discussion of jamming agents, despite many significant developments, note-level morphing is not at all apparent in this field, suggesting a possible niche application. Techniques for note-level morphing can be used to create new musical material for the jammer to output, whether morphing the realtime human input or a database of music. In this way, the *hybrid* aspect of morphing is more pertinent than the *transition*, although transitioning could also be applicable in situations where an "accompanist agent" is required to perform medleys.

3.3.3 Adaptive music

Adaptive music occurs when the interactive algorithm generates music in response to data that concerns a world-state, rather than musical or instructional input. It is perhaps the most commercially relevant field of algorithmic music practice, being applied primarily to the computer game industry (Neil 2005; Clark 2007), but is paradoxically one of the least developed in terms of algorithmic sophistication (Sanger 2003). That is, most adaptive music systems recombine sections of pre-produced electronic music, rather than utilising formalised musical knowledge to generate the musical backing.

The game *Need for Speed 3* (Electronic-Arts 1998) uses car speed to select the electronic loops with the appropriate intensity. Games such as *Frequency* (Sushi 2002) and *Amplitude* (Perry 2003) are especially interesting in that music is the central aspect of gameplay, however, even when this is the case, they do not go beyond the simple one-to-one mapping of button strike to playback that is typical in music games. Game music composers need to expend considerable effort in carefully crafting segments and transitions to suit the recombinant algorithms (Sanger 2003; Whitmore 2003; Apple 2006).

It is evident that individual game companies have tended to develop idiosyncratic low-level audio solutions in parallel for the platforms relevant to them (Sanger 2003), for example the *Miles SDK* (RAD 2004), *MusyX* (Factor-5 2000), *Audiality* (Olofson 2005) and many others that are reviewed by Brandon (2006). Despite this, there has been a slow movement towards more unified standards over the past few years (MMA 2003; Sanger 2003) and open discussion of adaptive music techniques is becoming more common (Paul 2003; Whitmore 2003; Apple 2006). Some historical context for game audio is given by Brandon (2004).

Currently, the most significant adaptive music tool is *DirectMusic Producer* which is detailed below.

DirectMusic Producer

DirectMusic Producer (DMP) is currently the most well developed, usable and freely available tool directed at composers who write adaptive music for computer games (Microsoft 2004). While many other engines exist, systems that aim to facilitate an interactive or continuous generative delivery are few (Factor-5 2000; Beatnik 2002; SSEYO 2004) and have less functionality. None of the algorithmic music techniques used by *DMP* are new; in fact they are quite rudimentary. The advantage of *DMP* is that the techniques are employed within a user-

friendly application which is integrated into a much larger game development Application Programming Interface (API).

While more comprehensive explanations are provided in *DirectX 9: Audio Exposed* (Fay, Selfon et al. 2003) as well as help menus from within the program itself, a short summary of the aspects most pertinent to adaptive music is provided here – specifiable random deviations in audio playback to overcome repetition; and a basic grammar for labelling and interpreting note sequence segments that includes the estimated aesthetic impact of patterns, the harmonic nature of a musical pattern and the musical role of the pattern.

DMP provides some functions that aim to alleviate repetition in computer game music – pitch, volume and duration variability of sound samples and pattern variations. Random variation of sonic parameters such as this is done much more extensively by *KOAN* (SSEYO 2004) and has existed within any programmable synthesiser for decades, not to mention the works of pioneers such as Xenakis (1992). Recombination of pattern variations is even older, the canonical example being Mozart's Dice Game (Roads 1996).

DMP enhances the effect of recombination through the labelling of "variations" with a musical grammar. Variations for patterns have a number from one to 32, and in this way can be ordered or shuffled on playback. They can also be flagged as being harmonically uncompliant with certain scales and chord-types, meaning that they have less (or no) chance of being selected when the specified harmonic configuration is playing. The level of detail available is quite extensive, incorporating many relevant grammatical terms from western music theory. It is also possible to specify what is harmonically required of the next (destination) chord in the sequence for the variation to be compliant.

"Patterns" (a *DMP* term, meaning a bundle of variations) can also be labelled to assist the recombination algorithm. The "groove" range is used to help determine which pattern should be playing when they overlap – when the groove level (an expression of emotional intensity) is outside the groove range assigned to a pattern it will not be played. As well as this, discrete musical roles such as "intro", "fill", "break", "end" and custom labels can be assigned to the pattern. However, apart from "intro" and "end", functionality that meaningfully interprets this labelling system is undefined by *DirectX* itself.

Overall, *DMP* is designed as a tool rather than a musical agent. The elements that involve stochastic generative algorithms could be lightly regarded as the **heuristic** compositional approach, however most of the musicality stems directly from the composer. The fact that *DMP* is currently the most highly developed tool available signals a niche that may be filled by techniques such as note-level morphing – an observation that is supported by Jonas Edlund's

recent efforts to apply note-level morphing in *The Musifier* computer game music engine (Edlund 2004). The goal here is for the game music composer to create passages that reflect the various moods in the game and for *The Musifier* software to create music at any particular point in time that expresses the precise combination of moods required by the world-state.

3.3.4 Interactive installations

Interactive installation covers a diverse range of approaches, all of which involve some form of physical and spatial interaction with a visual and/or sonic process. This field is in some ways more closely associated with new media visual art and design than algorithmic composition and is thus of only peripheral significance to the project. However, due to the potential for application of morphing to interactive installations, it has been investigated to a small extent. An overview of the field is given by Sommerer and Mignonneau (1998) while Krueger (1991) is of especially historical significance. John McCormack has applied sophisticated musical processes to interactive installations, involving evolutionary systems (McCormack 2003) in a mixture of the **abstract** and **trial** compositional approaches. Garth Paine is noted for development of installations based on physical gesture (Paine 2007) and establishing that it can sometimes be more appropriate to view the installation as a musical instrument rather than adaptive phenomena (Paine 2004). Most installations are fairly experimental and include interactive musical algorithms that relate to the **abstract** compositional approach.

Tabletop interfaces are particularly pertinent due to the topological affordance of morphing. The most comprehensive repository of information regarding current tabletop interfaces for electronic music has been compiled by Martin Kaltenbrunner (2006). This includes the *reacTable** (Jordà, Kaltenbrunner et al. 2005) which uses video tracking to locate special fiducial markers on a semi-transparent table that is also projected upon. The markers control various audio effects or generators within a patch. The *reacTIVision* software libraries (Bencina, Kaltenbrunner et al. 2006) that drive the camera tracking system were also used within this research to demonstrate the possibility of applying the morphing software to the interactive installation paradigm. The alternative to *reacTIVision* was the *Augmented Reality Toolkit* (Kato 2006) which was discussed with Rodney Berry who has used it within a number of projects (Berry, Makino et al. 2003; Berry, Makino et al. 2006). It was not investigated directly due to the increased time that would be required to develop customised *C* code.

Overall, interactive installations are particularly experimental, and most techniques with a degree of novelty are applicable to the field. This includes the note sequence morphing algorithms I have developed, as demonstrated by the *Morph Table* installation (see the conclusion).

3.3.5 Summary of interactive music

I reviewed a number of significant interactive music works within a range of application contexts – meta-instruments, jamming agents, adaptive music and interactive installations. Works that are particularly notable have been analysed in technical detail and this process was useful in providing inspiration for later developments of my own. For example, *The Continuator* showed how Markov techniques can be applied in realtime interactive contexts, *GenJam* highlighted the creative nature of genetic algorithms and *DirectMusic Producer* demonstrated that a surprising amount of adaptivity can be achieved with only basic techniques.

A number of opportunities for the application of note-level morphing have become apparent within each interactive music context, highlighting the potential usefulness for morphing to the whole field. This includes control of the morph index as a meta-instrument, hybridisation of source and target to generate material for jamming agents, seamless transitioning between musical states of computer games in adaptive music, or table-top interfaces for morphing in interactive installations. Having established the potential for morphing, the following section will review pervious attempts at note-level morphing.

3.4 Review of note-level morphing

A review will now be presented that examines instances of automated note-level morphing – the process of generating a **hybrid transition** between **source** and **target** note-level music. This last review positions my research within the narrowest possible context – viewing only the projects that share, at some level, the core focus of note-level morphing.

Five particularly significant works of note-level morphing were by Mathews and Rosler in 1967, Larry Polansky in the 1980s and into the 1990s, Horner and Goldberg in 1992, Danny Oppenheim in the mid 1990s and Jonas Edlund more recently from 2004. After reviewing the works of these investigators in some detail, other works, that either do not have note-level morphing as their primary subject or are less significant, are also mentioned so as to provide a greater sense of context.

The five primary works are compared in relation to the various significant research activities that they have engaged in. *LEMorpheus*, the system I developed, is contrasted to the other projects. The presence or absence of each research activity in each system is tabulated, clearly showing how *LEMorpheus* attends to numerous nascent research opportunities. The comparison was useful as an overview and guide to the research. It should not be construed as a criticism of the

earlier systems, as each was limited by the technological and aesthetic boundaries of its time. This is particularly apparent in the first system, *GRIN (GRaphical* INput).

3.4.1 GRIN

Max Mathews appears to have created the first musical morphing algorithm in 1966 on the MUSIC IV platform (Mathews and Rosler 1969). This work was developed as a demonstration of the algorithmic possibilities of Mathews' and Rosler's GRIN94 program. In this system, a monophonic melody was represented with separate functions, or envelopes, for each dimension of amplitude, frequency, duration and glissando. The frequency functions were made of flat (gradient 0) segments for the tone of each note, while the amplitude function was used to accent the first beat in each measure. Glissando was not used in the morphing example. The discrete note durations, or inter-onset times, required conversion to a continuous function in order to become algebraically manipulable.

Mapping discrete start times to a continuous domain is a problem that can be approached in many ways; Mathews' and Rosler's technique is particularly ingenious. To generate a melody, a note would be created and the duration function would be sampled at that point. The sampled value would specify the inter-onset distance to the next note and sample point. The "self-synchronising" form of the duration function required that each segment had a gradient of -1, so that if the sampling is ahead or behind a certain amount, the next note and sample point would be in time.

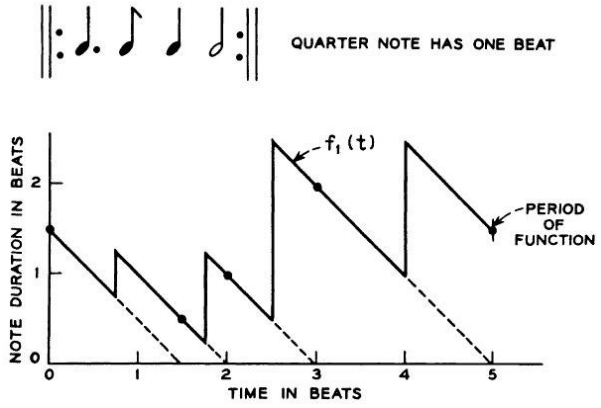

Figure 1 Self-synchronising function for inter-onsets (reprinted with permission).

Halfway between each note was chosen arbitrarily as the start and end points for each of these segments. Combining self-synchronising functions with others results only in other self-synchronising functions and so the coherent quantisation of durations to known values is

inherent in the style of representation. Having dealt with the problem of continuous representation, morphing becomes simply a matter of combining the functions in each pattern, using the morph index to weight each one.

The compositional approach employed by Mathews and Rosler appears to be **abstract**. It is approached as an exploration into what might occur when the music is parameterised and combined, as opposed to designing the algorithms strongly with preconceived notions of how it should sound. At the lower level, **analytic**, **transformational** and **generative** algorithms are all used. The conversion of note sequence to envelopes is analytic, the combination of envelopes is transformational and the rendering of the combined envelopes into a note sequence is generative. All of these operate within a narrow context, as no external or randomly accessed data is available to the algorithm.

For its time, a somewhat convincing result was recorded and produced on the vinyl disc accompanying the book (Mathews and Rosler 1969). It morphs from *The British Grenadiers* to *When Johnny Comes Marching Home* and back (~3.1).

3.4.2 HMSL

The *Hierarchical Music Specification Language (HMSL)* was developed by Phil Burke, Larry Polansky and David Rosenboom from 1980 and is implemented in *FORTH* (Burke, Polansky et al. 2005). It is partially inspired by the musical theories of Jim Tenney (Polansky 2006), including the notion that general patterns should be easily mappable to various levels of a music hierarchy. Polansky developed code in *HMSL* to aid the experimental morphing of music within some of his compositions, including *Distance Music* (Polansky 1987), *Bedhaya Guthrie/Bedhaya Sadra (Polansky 1996)*, *51 Melodies* (Polansky 1991), *Two Children's Songs* (Polansky 1992) and *Road to Chimachum*. MIDI renderings of these last three can be heard online (Polansky 2006).

This music was based on a theoretical framework developed by Polansky that explores and extends the application of mathematical set theory and similarity theory to experimental music. These ideas have been presented at conferences (Polansky and McKinney 1991; Polansky 1992) and covered more comprehensively in journal publications (Polansky 1996). To summarise the primary aspects, source and target are conceived as ordered sets. Given this representation, various analytical metrics can be applied to obtain some notion of distance between the patterns and, conversely, mutation algorithms can generate music at a specific distance (the morph index) between two sets. The various approaches are classified according to their foci and techniques: interval magnitude (difference between one item in the set and the next) or direction (up or down), linear (processing the set from start to finish) or combinatorial

(utilising intervals from each item in the set to every other item), unordered (non-structural statistics) or ordered (utilising the sequential order of the pattern). *HMSL* is unsupported by modern operating systems, although much of it has been ported to Java as the *Java Music Specification Language* (see below).

Overall, the compositional approach taken by Polansky is mostly **abstract**, as evidenced by the obviously experimental nature of his mappings and the music. Some works also showed the algorithmic modelling of musical stylistic intentions through **heuristics**, for example, *Bedhaya Guthrie/Bedhaya Sadra*. The piece *Drawing Unnecessary Conclusions* (Polansky 1987) utilised morphological metrics in **analytic** algorithms, and human performers as the **generative** component. Because there was an explicit criteria that the generated shapes be a certain metrical distance from each other, the compositional approach was somewhat related to **trial**, however there was no large population of candidates which are matched against the criteria. In this case, the approach to composition is ultimately carried out by the individual performer. An example of a harmonic morph created by Polansky using *HSML* is included (~3.2).

3.4.3 Horner and Goldberg

Horner and Goldberg (Horner and Goldberg 1991), published the first known application of GAs to morphing – probably also the first application of evolutionary computation to music (Biles 2008). Horner and Goldberg refer to the process as “thematic bridging”, where the source note sequence is transformed into the target via a sequence of operations, concatenating the result of each operation into a complete bridge. A fitness function appears to have been used at some stage to gauge the similarity of each bridge segment to the target, however it is not clear precisely where and how it was applied. Some aspects of fitness or selection appear to rely on user input, although the exact workings of the process are a little unclear as the results were only published in a short conference article. The music representation consisted of pitch and amplitude information. Horner and Goldberg’s software was used to compose a piece “Epistasis”, which I have unfortunately been unable to find.

3.4.4 DMorph

Daniel Oppenheim first published a short paper on morphing, presenting the *DMorph* software (Oppenheim 1995), which was implemented as a computer assisted composition tool within *DMix* (Oppenheim 1993) and discussed in detail as a patent (Oppenheim 1997). The algorithm was realtime, interactive and deals with **n-source** morphs, which extended the original definition of the morphing function to include any number (n) of input patterns. *DMorph* only morphed between single parts, however, as it existed within *DMix*, a multi-part environment, multi-part

morphs were possible. Despite the option of multi-part morphing in Oppenheim's system, there was no inter-part communication, for example, various parts following the same tonality. In the implementation of *DMorph*, four sources were used, each relating to a corner of a Cartesian plane called the *Mutant Ninja Tennis Court*. The morph index was a point on this two-dimensional plane. The emphasis was less on automatic transitioning (from source to target) and more on the creation of a musical hybrid through interactive topological navigation of the *Mutant Ninja Tennis Court*.

Oppenheim's procedure is to group notes from each source together based on a mapping that is selected by the user. Each group, which includes a note from source and target, relates to a note that will be generated during the morph. Once the notes from source and target are grouped in this way, the note properties of all notes in the group are interpolated and this value is used to create a new note. There are two different generic implementations of this procedure - *time warped* grouping creates groups based on the order of the notes, while *time synchronous* grouping creates groups based on the distance of note onset. The decision to lock, interpolate, or apply weighted selection for individual parameters such as pitch or onset is made by the user.

The compositional approach used in *DMorph* is **abstract**, based on interpolation of musical parameters and exploration of the topology, rather than striving towards any explicit or implicit musical goals. The grouping process is **analytic**, the weighted combination of parameters is **transformational** and the "note-groups to note-sequence" process is **generative**. The fact that the note grouping algorithm has access to all of the notes from source and target means that there is a moderately wide breadth of context.

Musical demos are no longer available from IBM's website, however, Oppenheim was gracious enough to send them by email (Oppenheim 2006) and some examples are included (~3.3, ~3.4, ~3.5).

3.4.5 The Musifier

Jonas Edlund has developed an adaptive music system, *The Musifier* (Edlund 2004), which utilises note-level morphing as a key component. Edlund presents *The Musifier* as an 'orchestra-pit' agent – fulfilling the role of a theatre's orchestra pit, but in the context of computer games. *The Musifier* performs **n-source** morphing on different themes provided by a human composer. The themes are linked to particular dynamic parameters within the game state, for example the health-level of the player character, the position on the map, the proximity of non-player characters or enemies. The human composer designs the musical theme to reflect the emotional significance of each game state parameter and *The Musifier's* morphing algorithm hybridises

them into the most appropriate music, in realtime, based on the value of the game-state parameters.

The details of the morphing techniques that Edlund uses are a trade secret, however musical demonstrations are available for download. More recently a web application has been made which allows a user to specify the weights of different themes (Edlund 2006). A particularly useful technical advance is apparent simply through listening to the examples – the problem of morphing between parts of different timbre has been adequately handled in MIDI by cross-fading the volume of parts on two different channels and sending identical note events to both channels. Through personal communications with Edlund, my initial speculation of an abstract harmonic representation to provide unified movement to harmonic parts has been confirmed. Rhythmic segments appear to be treated as indivisible gestalt units. As well as this, melodic patterns sometimes also appear indivisible, switching to the new theme at an appropriate cross-over point.

Edlund uses three criteria for adaptive music and morphing – responsiveness, continuity and complexity. Responsiveness is how well the system responds to change. Continuity is concerned with matching the contour of the changes and changing smoothly. Complexity is how well the algorithm can convert the many dimensions of game state data into an equal number of dimensions of musical data and then into suitable music. For most examples, *The Musifier* appears to do quite well according to these criteria, although without rigorous testing it is difficult to comprehensively ascertain the various musical successes and aesthetic traits of Edlund's work.

The Musifier most likely reflects the **heuristic** approach to composition, however, it is difficult to classify the compositional approach without viewing the algorithms. The musical intentions are fairly clear, as evidenced by the adherence to popular, tonal music styles and the criteria for adaptive music and morphing that were motivated by the strong commercial imperative. Because of this, the algorithms are likely to be based on the **heuristic** or **trial** approaches, rather than **abstract**. The algorithm does not appear to require significant computation in order to create the morph, which implies a fast **heuristic** over the often more computationally intensive **trial**. An example is included with source (~3.6), target (~3.7) and morph (~3.8).

3.4.6 Others

A number of algorithmic music systems, other than those examined in detail above have applied morphing or morph-like techniques. Momeni and Wessel developed the *Beat-Space* software using *MAX/MSP* which morphs between parameter states on a 2D surface. Gaussian kernels are used to control the prominence of each parameter state on the surface (Momeni and Wessel

2003). While the software was primarily concerned with morphing sonic parameters, the *Beat-Space* component dealt with musical material, morphing between parameters that control probabilities of beat generation within eighth-note slots. The source and target are deterministically represented such that the probability for any slot can be only zero or one, while the morph is generated from the non-deterministic interpolations.

KOAN (SSEYO 2004) can also perform note-level morphing, as boasted in *Presswire* (M2-Communications 1997):

> *"Examples of just some of what the IKMC (Interactive KOAN Music Control) can accommodate dynamically in real-time include: addition or alteration of melodic or rhythmic patterns, smooth morphing between two KOAN pieces, changing of patches, application of filtering effects to change the sound palette, modification of auto-chording, alteration of the generative rules underpinning the piece, deletion or addition of KOAN player 'voices' and rules, and semi-interactive MIDI file playback."*

The approach is similar to Momeni and Wessel's, but deals with pitch as well as rhythm. *KOAN* was designed primarily for websites and hand-held music making devices with limited storage space and the requirement for constantly changing music over long periods.

Nick Didkovsky and Phil Burke (2001) have extended the capabilities of *JMSL* beyond the original *HMSL*. Particular aspects of *JMSL* which are relevant to morphing are the *Binary Copy Buffer Transform (BCBT)* (Didkovsky and Burke 2004) and an applet called the *Schubert Impromptu Morpher* (Didkovsky 1997). The *BCBT* is a function that is part of the score editing window in *JMSL* (Didkovsky and Burke 2004), where the user can copy segments of music into two different buffers. The *BCBT* function then uses a morphing algorithm to combine the two buffers and paste the result onto the score. In this way, buffer one is source, buffer two is target, and the pasted result is the morphed hybrid. The *Zipper Interleave Transform* is a morphing algorithm that comes with *JMSL* which iterates through source and target, alternately placing an element from one or the other into the morph. Through the extensible code design there is great potential for users of *JMSL* to create custom *BCBT* plug-ins for the score editor, however this did not appear to be a particularly active area of development. Didkovsky's *Schubert Impromtu Morpher* applet stochastically generates music from statistics obtained by analysing a Shubert performance, as source. The target is specified from user-defined statistical values. The user controls the morph index, and can disable the interpolation of individual statistical parameters (Didkovsky 1997).

As well as being a DJ agent (see section 3.2.3), Tristan Jehan's *Skeleton* software can automatically create hybrid tracks, from source and target audio (Jehan 2005). The algorithm

extracts a music structure (note-level, metre) description from the target, and “fits” audio snippets from the source into it, as per concatenative synthesis (Schwarz 2004; Sturm 2004). Jehan describes the process as being cross-synthesis on the level of musical structure.

In further relation to sound morphing, Ircam’s software *Diphone* (Rodet and Lefevre 1997) also uses an analytical and concatenative approach to morphing. Polansky and Erbe (1996) applied morphological metrics to spectral mutation in the *Sound-hack* software. Paul Lansky designed custom sound morphing algorithms for the composition *Quakerbridge* (Oppenheim 1997) and Cook has morphed with vocal synthesis algorithms (Cook 1998).

Robert Winter has developed a system in *Pure Data* that deals with interpolation of musical expression and structure, with a primary focus on emotion (Winter 2005). Canazza (2001) also developed a system for morphing musical expression. Algorithms that deal with note-level representations do not seem apparent. Stephen Ingham (Ingham 1996) has developed software to interpolate values in Standard MIDI Files in the *Patchwork* environment. Christian Renz has developed an application in *Perl* called *MIDI Morph* (Renz 2005), designed for note-level morphing. It is currently alpha and not in active development. Berger (1995) submitted an abstract with plans to develop a note-level morphing system.

There are no doubt other small scale algorithmic music systems that relate to morphing, indeed, any parametrically well-defined problem space could be the subject of morphing through interpolation. However, enough has been presented to sufficiently express a sense of the surrounding context.

3.4.7 Summary of opportunities for note-level morphing research

Through a comprehensive review of note-level morphing systems, only four works with a significant degree of relevance and capability have been found, highlighting a potential for growth in the field of note-level morphing. Examining these works in detail, I have identified a number of research niches that have not, until now, been thoroughly explored in the context of note-level morphing. These opportunities, which will be explained below, include: modelling the **trial** and **heuristic** approaches to composition, rigorous testing, musical coherence within a MEM context and contemporary computing platforms.

A large niche for research in note-level morphing is in modelling the **trial** approach to composition, through evolutionary algorithms. While the processes used in *The Musifier* are a trade secret, they appear from my observations to be based on the **heuristic** approach. The

other existing systems – *GRIN, HMSL* and *DMorph* – are all based on musical **abstraction**. Horner and Goldberg modelled a **trial** approach, however, while this work was highly novel at the time, the lack of detail, scope and application allows considerable room for additional developments.

Another opportunity for new research in morphing appears to be formal testing, which is entirely absent from all of the projects that were examined. Some researchers/projects utilised informal feedback, for example, Edlund released beta versions of *The Musifier* with the intention of gaining feedback and Polansky had access to reviews and personal opinions regarding how his pieces were received. However, none of the projects utilised systematic musical evaluation from a group of users or listeners. As a result there was no collection and interpretation of qualitative data, which could then lead to further refinements of the algorithms. If such methodologies had been implemented, there would be impetus towards music that was considered by survey participants – and hopefully most others – to be of a high standard.

Listening to the musical output of the various systems also presents an opportunity – the possibility of morphing algorithms which appeal to the popular sensibilities of MEM. It is apparent from listening to the *When Johnny Comes Marching Home* to *The British Grenadiers* morph created on *GRIN*, that it is not passable as marching music. As well as this, the work itself predates MEM as it is known today. Polansky's morphing compositions using *HMSL* were appropriate for the intended context of avant-garde music, however the algorithms are unlikely, without significant extraneous production work to be successfully applied to MEM. Either way, application to MEM, as a research opportunity, was not investigated by Polansky. *DMorph* was tested on musical styles such as Latin and Classical. None of the demos provided by Oppenheim were in the MEM genre and, without access to the software, it is difficult to claim with any certainty for or against the applicability of *DMorph* to MEM. Nevertheless, it is clear that the application of note-level morphing within a MEM context has not been thoroughly investigated by Oppenheim. *The Musifier* is designed to cater to computer game music and therefore is partially situated within a MEM context, as with *LEMorpheus*. It is difficult to objectively gauge how *The Musifier's* music might be received without tests; however, from the musical demos, it appears to afford music with extended chord progressions and simple rhythms. This is contrasted to *LEMorpheus* which is applicable to short chord progressions and complex rhythms.

The ability for multiple parts to work together, sharing data during the morph, is another aspect of note-level morphing that is in need of further research. Only LEMorpheus and *The Musifier* address this. *GRIN* was monophonic, *HMSL* dealt with multiple voices in ways that were specific to the individual compositions and there are no morph examples from *DMorph* between a source

and target with more that one part in each. Multiple parts could be used through *DMix*, which contained *DMorph*, however, the morphing process is parallel for each of those parts; that is, no shared inter-part information. Horner and Goldberg computed the layers separately.

In his patent, Oppenheim (1997, p 42) recognised how useful it could be to assign each part a separate musical function – for example, bass, lead, drums – and use this information to inform the morphing process. *The Musifier* explicitly includes functional parts: the abstract chord and scale sequence part, bass, melody, chords and percussion. I also addressed part function, but in a more flexible manner - various settings of *LEMorpheus* parameters afford the morphing of different part functions and I have saved particular settings of parameters and applied these to particular part functions as necessary. Oppenheim (1997, p 42) also pointed out the need for the user to be able to morph each layer independently of others. It is notable that *LEMorpheus*, in the *Morph Table* mode, is the only system where this is currently possible.

Exploring new ways for morphing algorithms to be interactive is another aspect where research opportunities become apparent. *GRIN* and *HMSL* are not realtime. *DMorph* and *The Musifier* both allow for realtime modification of morph index for each of their sources. *DMorph* is perhaps the most advanced in terms of interactivity, being able to independently morph musical aspects such as rhythm and pitch on the *Mutant Ninja Tennis Court*. However, no algorithmic designs for note-level morphing that suit interactive interfaces beyond the mouse and screen have been explored. *LEMorpheus* can operate in *Morph Table* mode where parts are morphed independently on a physical tabletop interface.

In situations where the source and target have contrasting timbres, morphing on the note-level initially presents an obstacle to integration of timbre – program changes are inadequate due to slow speed and the occurrence of stuck notes on some synthesisers. The inadequacy of timbre integration in MIDI morphing was mentioned by Oppenheim (1997). Both Oppenheim and Edlund overcame this by utilising two MIDI channels, one for source and one for target. Inspired by this, I developed a similar scheme in *LEMorpheus*. In personal communications, Oppenheim also highlighted the use of system exclusive MIDI messages (2007) as another potential method to control the timbres.

Finally, only *LEMorpheus* and *The Musifier* are executable on contemporary computing platforms (post Windows 98 and Mac OS 9). While this is not particularly relevant to algorithmic music techniques it is worth noting, as it hinders other researchers from replicating the results.

The opportunities are summarised in the following table:

Research Opportunity	GRIN	HMSL	H&G DMorph	Musifier	LEMorpheus
Musical trial approach			Y		Y
Musical abstraction approach	Y	Y	Y	Y	Y
Musical heuristic approach				Y	Y
Formal Testing					Y
Mainstream Electronic Music				P	Y
Independent part morphing					Y
Inter-part communication				Y	Y
Functional parts				P	Y
Realtime morphing			Y	Y	Y
Interactive graphical interface	Y	Y	Y	Y	Y
Interactive physical interface	Y				Y
Morph between MIDI instruments			Y	Y	Y
Contemporary operating system				Y	Y

Figure 2 **Summary of research opportunities and the four major note-level morphing systems. "Y" indicates that the opportunity was investigated to a significant extent by the system, while "P" indicates that the opportunity was investigated partially. Blank indicates that no investigation has occurred. H&G stands for "Horner and Goldberg".**

As evident from the table, my study addresses a range of research niches that have not been thoroughly explored in the past. These are now summarised. *LEMorpheus* includes three different algorithms that partially explore the compositional approaches of abstraction, heuristics and trial. The latter two were the subject of formal tests, firstly a focus-concert and then an online survey. The MEM music created by *LEMorpheus* was found to be applicable to mainstream music contexts such as computer games and live electronic music performance. The *LEMorpheus* system allows for any number of layers, with a separate morphing algorithm and settings specific to the musical function of that layer. When applied to the physical interface of the *Morph Table*, each of the parts have a separate morph index. *LEMorpheus* allows any part to follow the tonality of any other part. *LEMorpheus* operates in realtime and is able to handle morphing between two different MIDI instruments. Lastly, *LEMorpheus*, being written in *Java*, is able to be tested with relative ease.

Having contextualised my own research within the fields of algorithmic composition, interactive music and note-level morphing, I shall now provide a thorough explanation of the techniques I used, beginning with the system infrastructure that is behind *LEMorpheus*.