Measurement of similarity in music: A quantitative approach for non-parametric representations.

Measurement of Similarity in Music: A Quantitative Approach for Non-parametric Representations

Keith S. Orpen & David Huron [1]
Computers in Music Research, Vol. 4, (Fall 1992), pp.1-44.
A one-page overview/summary of this document.

Introduction

Many of the stock problems in music analysis center on the recognition and identification of patterns. Such patterns may include simple features (such as chords, motives, or formal recurrences), or more complex patterns (such as melodic processes, stylistic traits, and other features). Computers are well-suited to tasks such as exact pattern matching. In computer-based pattern matching, a source pattern (or template) may be defined, and occurrences of the pattern sought in some target document. Although pattern matching is important in musical applications, exact pattern matching has a limited utility. In music, there are relatively few instances of exact recurrences of a given pattern. Themes may appear in various guises, rhythms may be modified, melodies are embellished, chords are voiced differently, harmonic substitutions occur, changes of instrumentation are made, and repeated passages (such as recapitulations) often appear in different keys.

When patterns are inexactly repeated, one way of identifying instances of a pattern is by defining a pattern *grammar*. An example of a pattern grammar would be the pattern: "X" followed by either "Y1" or "Y2," followed by anything, followed by "Z." [2] Although pattern grammars permit much greater flexibility in the defining of patterns, the process of pattern matching produces only one of two outputs: for a given input, either the pattern is present or it is not. In contrast to the concept of pattern matching is the concept of similarity. Rather than identifying whether two inputs are equivalent (according to some grammar), we may be interested in characterizing their degree of similarity. In formal terms, the

difference between similarity and equivalence lies in the property of *transitivity*. In the case of equivalence, if *a*=*b* and *b*=*c*, then *a*==*c*. In the case of similarity however, if *a* is similar to *b* and *b* is similar to *c*, it does not necessarily follow that *a* is similar to *c*. Equivalence-relations are transitive, whereas similarity-relations are intransitive.

In musical applications, similarity may be more important than pattern matching for two reasons. First, many aspects of musical experience appear to be dominated by impressions of *resemblance* rather than impressions of discrete identity or *equivalence*. Second, although exact transformations are frequently used in some types of composition (notably serial and minimalist music), most music-making employs much less formal manipulations of material. Compositional methods appear to place greater emphasis on (inexact) variation of materials rather than on exact formal transformations. When serial methods of analysis are applied to non-serial musics, by way of example, serial concepts of identity often prove unduly restrictive. A more musically-pertinent analytic approach might seek to characterize the nature of musical resemblances.

Similarity: Qualitative and Quantitative Aspects

When are two things similar? The question is trickier than it may seem at first. Is a crow similar to a diesel locomotive? In comparing the similarity of two entities, two broad questions may be posed: (1) in what way are two things similar? and (2) to what *degree* are they similar? We will refer to these two aspects of similarity as the qualitative and quantitative aspects respectively. Notice that the qualitative question is logically prior to the quantitative question. That is, we cannot characterize the degree of similarity until we have identified some qualitative dimension by which the entities may be compared. In the measurement of similarity, the analyst's first task is to identify some property or properties for comparison. Consider, for example, the word/concept "butter." In what ways can something be similar to butter? Margarine is similar *in taste* to butter. The spoken word "batter" is similar *in sound* to the word butter. Grease is similar to butter in *viscosity*; cheese may have a similar *nutritional* value to butter, and so forth. In short, butter is similar to margarine, batter, grease, or cheese -- but it is similar to these other things in different (qualitative) ways.

When we say simply that "two melodies are similar," we are using an unconscious short-hand, in which we neglect (or are unable) to identify the specific qualitative dimensions according to which the melodies are "close." Qualitatively, melodies may have similar pitch contours, similar structural tones, similar rhythms, similar harmony; they may both evoke a similar mood; they may both express similar themes such as unrequited love, shame, or happiness; they may both be especially quiet, or simply have a similar duration. Both melodies may be strophic in form, or both may address a similar audience (e.g., children). Like the crow and the locomotive, whether or not we regard them as being similar depends upon

our qualitative frame of references: the crow and the locomotive may be dissimilar in size, but similar in color -- they may differ in power, but both the crow and the locomotive may provide similar literary metaphors.

Having identified some qualitative property by which two things may be deemed similar, we may then attempt to characterize their quantitative similarity or degree of closeness. In some cases, a quantitative scale already exists making it possible to characterize directly the quantitative similarity for a given qualitative property. For example, a consumer may recognize two items costing $9.29 and $9.70 as being more similar to each other than to a third item that costs $12.80. In short, the qualitative property of "cost" provides a ready-made quantitative scale for characterizing nearness. In other cases, we may have a choice of scales, as when we say that 500 grams is similar to a pound. Other qualitative dimensions are less immediately characterized quantitatively. For example, having identified that "butter" and "batter" are similar in sound, we can note that the word "putter" is more similar to "butter" and the word "matter" is less similar. However, unlike cost or weight, there is no obvious quantitative yardstick by which we can make these comparative claims. The absence of a ready-made yardstick is inconvenient, but it is not necessarily problematic. In real applications we can often invent an appropriate yardstick, although the resolving power of such a yardstick may be quite crude. For example, in the case of similar-sounding words, we might carry out perceptual experiments that provide some quantitative data indicating (say) the relative frequency of confusion between various phonemes.

For quantitative data, a number of numerical and statistical methods have been devised as measures of similarity. For example, Pearson's coefficient of correlation [3] provides a useful way of measuring the similarity of the rise and fall of two sets of numerical values. If we wished to determine whether the annual pattern of precipitation in Montréal is more similar to that of Melbourne, or of Miami, we could align the monthly precipitation data and calculate Pearson's coefficient of correlation. We would find that Montréal correlates most strongly with Miami. But we must be careful of the qualitative assumption made in any quantitative measure. Since Melbourne is in the southern hemisphere, correlating the calendar year precipitation pattern is questionable. If we were to shift the Melbourne data by six months, we might find that the resulting correlation with Montréal would be stronger than the original correlation with Miami. In short, the quantitative results depend upon the qualitative decision to measure the *seasonal* precipitation patterns, rather than the *calendar year* precipitation patterns.

This example underscores an important point about the relationship between the quantitative and qualitative aspects of similarity. In measuring the similarity between two things, we can never be sure that there isn't some other (qualitative) dimension by which the two things exhibit a greater (quantitative) similarity -- relative to some set of things. In some analytic tasks we may be most interested in determining which elements of a given set are most similar according to a pre-

established qualitative dimension. In other tasks we may be interested in determining which qualitative dimension reveals the greatest similarity between two things. Not all data is quantitative in nature, so it is not always possible to apply parametric measures of similarity such as Pearson's correlation. Although many musical parameters may be represented quantitatively, it is not always possible to cast musical elements according to some quantitative yardstick. Often the information is in the form of discrete categories that cannot be ordered -- as, for example, in the case of different forms of notated articulation marks. In the case of non-quantitative data, an alternative way of calculating the degree of similarity between two things is to ask: how much "tinkering" is required in order to reach identity? What must be changed in order for "X" to be identical to "Y" and how extensive are the required changes?

The Measurement of Similarity for Non-Quantitative Data

In the field of computer science, the problem of similarity for non-quantitative data has been traditionally stated in terms of strings. A *string* is simply a finite sequence of characters drawn from some alphabet. An alphabet could be any collection of symbols, but in real computing environments, the alphabet is normally drawn from a set of pre-existing characters, such as the common *ASCII* set. These characters may, in turn, be used to represent elements in some existing symbol system. The alphabet may be finite, but it doesn't have to be. Innumerable systems of music representation have been devised in which score information is rendered as *ASCII* "strings." Some of these representation schemes employ finite alphabets (or signifiers), whereas other schemes employ infinite alphabets (see for example Huron, 1992). By representing music as strings of discrete signifiers, the problem of musical similarity can be expressed more generally as the problem of finding a good measure of similarity between non-quantitative strings.

Although two strings may represent non-quantitative information, it may still be possible to characterize the degree of similarity in a quantitative manner. One approach to this problem is suggested by research in approximate string matching. Approximate string matching is often described as the process of finding "near misses" to a given string in a list of known strings (such as defined in a dictionary). The most common applications of approximate string matching are in such tasks as recovering from typographical errors or spelling mistakes in language text. Inherent in the very idea of a "near miss" is the concept of "distance" between strings. There are many possible ways of defining a rule for computing a "distance" between two strings; different rules will lead to different estimates of similarity.

Edit Distance: The *Damerau-Levenshtein* Metric

One of the most prevalent and intuitively appealing approaches to measuring quantitative similarity is to calculate the *edit distance* between two strings (Damerau, 1964; Levenshtein, 1966; Ullman, 1977). Briefly, the edit distance between two strings can be defined as the minimum number of basic modifications (insertions, deletions and substitutions) that must be performed on one string (*source string*) in order to make it identical to a second (*target*) string. Performing an insertion means augmenting the source string by adding a symbol, whereas a deletion means removing a symbol from the source string. A substitution is the replacement of a single symbol in the source string by another symbol, which could be the same, or different. If a replacement symbol differs from the symbol it replaces, the substitution is called a *dissimilar substitution*. For each type of edit operation we may define a numerical penalty representing the magnitude of the modification. For example, the operations of insertion and deletion might be defined as adding a nominal value of +1 to the edit distance. A substitution might be defined as adding a value of +1 if is dissimilar, and zero if it is not. A dissimilar substitution is logically equivalent to a deletion followed by an insertion, so if we assigned an edit-distance penalty of +2 rather than +1, then the substitution operation would be redundant.

Whether or not a dissimilar substitution ought to be distinguished from a deletion+insertion depends on the qualitative notion of similarity. Qualitative similarities will depend on the field of application. In the case of typing, we commonly regard a word with one inccorect letter (e.g. "resesrch") as containing a "single" error. That is, we don't regard the error as arising from having typed the correct letter, deleted it, and then inserted an incorrect letter in its place. Although the "single error" scenario makes sense for typing, in another application, we might want to regard dissimilar substitution differently. If *S* and *T* are any strings, let *d(S,T)* denote the edit distance between them. It is easy to see that if *U* is any third string, then

d(S,T) = 0 if and only if S=T,
d(S,T) = d(T,S), and
d(S,T) <= d(S,U) + d(U,T).

These three properties are precisely what are required of a function in order that it be a so-called *distance function* or *metric*. The last formula is the "triangular inequality," and is the mathematical statement analogous to saying that flying from Los Angeles to Bangkok can cover no more distance than flying from Los Angeles to some third location and then to Bangkok, for example. When such a distance function or metric is used to calculate the edit dstance between two strings, it is referred to generically as a Damerau-Levenshtein metric (Hall & Dowling, 1980).

Notice that several variations on the edit distance are possible. We are free to define both the types of permissible edit operations and the numerical penalties associated with each operation. For example, we could define a single operation

that reverses the order of two adjacent characters (e.g., `ba' rather than `ab') and define the corresponding distance penalty as +1. Notice that if we assigned a penalty of +2, the edit distance would be the same as two dissimilar substitutions and so the "character reverse" operation would be redundant. Once again, the significance of an edit operation and the value of its corresponding penalty value depends on the field of application. In the case of typing errors, reversing the order of two adjacent characters is very common, and so two strings that differ by only a reversal of two adjacent characters would be considered much closer than two strings that have random differences in two adjacent symbols. As we have noted, we could modify the edit-distance penalties associated with each basic edit operation -- depending on the application. However, for simplicity, in the ensuing discussion we will assume all edit operations to have a penalty value of +1. Notice that we do not have total freedom in assigning the edit-distance penalties. The penalties must form a coherent metric. For example, we cannot assign a penalty of +3 for a substitution and +1 for an insertion or deletion without violating the triangular inequality.

Later in this article we will present several applications of this approach to problems of musical similarity. However, before exploring such musical applications, we will explain in greater detail some of the technical issues and assumptions involved in calculating Damerau-Levenshtein edit distances.

Computing the Edit Distance

In order to calculate the edit distance between two strings we must find a sequence of operations that exhibits the minimum possible edit-distance penalty. One way of visualizing the problem is illustrated in Figure 1. Suppose that we wish to find the edit distance between the strings "has" and "this," using only the operations of insertion, deletion, and substitution. The diagram in Figure 1 represents all possible edit operations that will transform one string into the other.

Figure 1. Computing the edit distance between the strings "this" and "has." Each arrow represents a basic edit operation and incurs a penalty of +1. Horizontal and vertical arrows represent insertion and deletion respectively of letters in the string "this." Diagonal arrows represent either a dissimilar substitution (+1 penalty) or equivalence (no penalty).

Every path following the arrows from the top left corner to the bottom right corner represents a possible edit sequence that will transform the string "has" into "this." Each arrow represents a basic operation; horizontal arrows represent insertion of the corresponding letter from the string "this" whereas vertical arrows represent deletion of a letter from the original string "has." All of the horizontal

and vertical arrows have a penalty or cost of +1 (which is not displayed). Each diagonal arrow represents either a dissimilar substitution (in the case where the corresponding letters from each word are different), or "doing nothing" (in the case where the corresponding letters are the same). The cost of a dissimilar substitution is +1 in this example. The diagonal edges are labelled with their respective costs. The number at each node (intersection) represents the cost of a lowest-cost path to that node from the top left corner, where the "cost" of a path is simply the sum of the costs of the arrows defining the path. As an extreme example, the path that traverses across the top-most row and then down the right-hand edge has a total cost of +7 -- corresponding to inserting the four letters of "this," followed by deleting the three letters of "has." Since the two words share two letters in common, this sequence of edit operations is wasteful. The path around the perimeter of the matrix is obviously not a lowest-cost path. The shortest path is seen to correspond to inserting a "t" to get "thas," leaving the "h" unchanged, substituting an "i" for the "a" to get "this," and leaving the "s" as is. This edit path has a total cost of only +2. In this example, there is only one minimum-cost path, but in general there may be several such paths. In most applications, it is only the edit cost that is of interest, and so the path itself is unimportant. From a computational point of view, finding a lowest-cost edit path grows exponentially as the length of the strings is increased. For strings of fifteen letters, calculating every possible path would be prohibitively time-consuming -- even on a very fast computer. Fortunately, there is a short cut that can speed up this computation considerably (Wagner, 1972).

It is clear that the lowest-cost of a path to a node on the left-hand edge of Figure 1 is simply the number of arrows that must be followed to reach the node, since all the vertical arrows on the left-hand edge have cost +1. Hence the lowest-cost to reach each node on the left edge is easy to compute, and can be stored in an array for future reference. In fact, moving across the diagram one column at a time, the lowest-cost of a path to each node in the current column can be deduced quickly from information stored in the array. Let us establish the convention that the "cost" of a node in a diagram such as Figure 1 is the cost of a lowest-cost path to the node from the top left node. If we imagine scanning each column from top to bottom, then the cost of a lowest-cost path to a given node is the lesser of:

- the cost of the node above it, plus one for an insertion
- the cost of the node to the left, plus one for a deletion
- the cost of the node diagonally above and to the left, plus one for a dissimilar substitution.

It is to be understood that in the case of an edge node some of the above neighbors may not exist and then the cost is the lesser of the costs computed from the remaining neighbors only. For the top left node, no such neighbors exist, hence it is assigned the cost of zero.

By mathematical induction it can be shown that if the costs of nodes on the left-

hand edge are computed correctly, and if *assuming* that the costs of the previous column are correct implies that the costs of the current column are computed corectly, then the cost of every node will be correct. We have already seen that the costs along the left edge are easily computed. The second premiss above -- that the results of the current column will be correct *if* those of the previous column were correct -- is easily verified. All arrows in the diagram point either to the right, downward, or "southeast." This means that at any given node, *all* paths to that node must approach via a node that is either in the previous column, whose cost we have assumed to be correct, or is above the given node and in the same column. In particular, any lowest-cost path must proceed via a node from the previous column or from above. Scanning the columns from top to bottom means that the cost of the node directly above the current node is known; indeed, it was the cost last computed. It is clear that if the cost of one of these three neighbors is known, then the cost of a lowest-cost path *that is constrained to pass through that neighbor* is given by the cost of the neighbor node plus the cost of the arrow connecting it to the current node. Since every lowest-cost path must in fact pass through one of the neighbor nodes, the overall lowest-cost is just the minimum of the costs through each neighbor -- so the three-part rule given above is valid.

Note that it is not generally necessary for the program to store explicitly the entire table above; if the metric is as simple as the one above, then the program need not store any more than one column at a time. The amount of storage depends on the degree to which the defined edit operations in the metric are "local" in their effect. In the above example, the metric is highly localized. The cost of every arrow depends on zero or two letters; either the arrow is an insertion or deletion and has a penalty of +1, or else it is a substitution and depends on whether the corresponding letters from each word are the same or different. Metrics that permit more elaborate types of string modifications may require larger pieces of the table to be stored at one time.

In musical applications we might want to assign a lower penalty to the insertion or deletion of a *repeated* symbol -- such as a repeated note. In this case we must store an array representing the entire diagram left of the current column in order to cover the possibility that the note has been repeated in every column. This is an example of a "non-local" metric since it relies on comparing symbols that could be arbitrarily far apart in the same string.

The *Simil* Program

The Damerau-Levenshtein metric described above was implemented by the first author using the YACC parser generator, the LEX lexical analyzer generator, and the C programming language (Aho & Johnson, 1974; Kernighan & Ritchie, 1988). The resulting program is called ***simil***. The input and output syntax for *simil* are designed to conform to the ***humdrum***. music representation system devised by the second author (Huron, 1988, 1991, 1992). Humdrum allows musical

information to be represented in a computer file, however humdrum does not specify how a given musical symbol or parameter should be represented. Humdrum provides a general framework within which symbol schemes may be defined and coordinated. This generality accommodates different ways of charcterizing musical information. For example, strings may represent absolute pitches, relative pitches, pitch classes, tablatures, semitone intervals, diatonic intervals, pitch contours, nominal durations, relative durations, simultaneities, chords, harmonies, dynamics, lyrics, etc., or any combination, subset, or extension of the above.

Humdrum data is encoded much like a two-dimensional table with columns and rows. Each table entry is referred to as a *data token*. Successive columns are referred to as *spines*, while the rows are called *records*. Each successive record (line of data) pertains to a given moment (or span) of time; hence time "runs down the page," so that subsequent records refer to later events. Each spine (column) in a humdrum file records information about some music-related property or attribute. Thus, a spine could correspond to a single voice in a polyphonic work, the lyrics of a song, MIDI keypress data, a Schenkerian middle-ground analysis, or some other musically pertinent information. Each spine is labelled with an "interpretation record" (marked by two asterisks) that identifies the type of encoded information. Normally, these interpretation records act as flags that invoke appropriate types of manipulations or processes that may be legitimately applied to a given data set. All humdrum tools are able to read files that contain any mixutre of interpreted spines, and organize their own output into spines so that the output may be used as input for other analysis tools. By way of example, consider the following trivial humdrum representations. In each of the following examples two spines are represented -- the left column represents an A major ascending scale, whereas the right column represents a concurrent A harmonic minor ascending scale. In the first example, the pitches are represented using ANSI standard pitch notations.

**pitch	**pitch
A3	A3
B3	B3
C#4	C4
D4	D4
E4	E4
F#4	F4
G#4	G#4
A4	A4

We might alternatively represent these scales according to integer values that represent the number of semitones with respect to middle C (=0). Notes above middle C receive positive integers, and notes below receive negative integers.

Once again, the left-hand spine represents the notes of A major ascending scale, whereas the right-hand spine represents the notes of A harmonic minor scale.

**semits	**semits
-3	-3
-1	-1
1	0
2	2
4	4
6	5
8	8
9	9

Alternatively, we could represent just the diatonic letter names or scale degrees:

**diaton	**diaton
A	A
B	B
C	C
D	D
E	E
F	F
G	G
A	A

Or we could represent these inputs as pitch-classes:

**pc	**pc
9	9
11	11
1	0
2	2
4	4
6	5
8	8
9	9

Or as a sequence of melodic intervals:

**mint	**mint

+M2	+M2
+M2	+m2
+m2	+M2
+M2	+M2
+M2	+m2
+M2	+A2
+m2	+m2

Or as changes of melodic semitone:

**melsem	**melsem
+2	+2
+2	+1
+1	+2
+2	+2
+2	+1
+2	+3
+1	+1

Notice that in selecting different forms of representation, we are, in effect, viewing the scales according to different qualitative dimensions by which quantitative measures of similarity are made.

How *Simil* Processes Data Tokens

In our earlier exposition of approximate string matching, we assumed that each character represents a single symbol. Using ASCII characters in this way limits the size of the alphabet of signifiers to the finite number of ASCII characters. In principle, the Damerau-Levenshtein metric is capable of handling alphabets of arbitrary size, so there is little reason to limit users to signifier sets no larger than the ASCII set. The *simil* program avoids this limitation by treating humdrum data tokens (typically groups of characters) as atomic signifiers or elements of the alphabet. In short, each humdrum data token is treated as a single signifier, no matter how may ASCII characters may be contained in the token. Thus, for example, the ANSI pitch representation "C#4" can be treated as a single symbol, despite the fact that it is made up of three characters. With the *simil* program this is achieved by assigning integers to each data token in such a way that two data tokens will have the same integer representation if, and only if, they contain identical text. By way of example, the following pair of input spines:

**text	**text
This	is

not	a
message	.
.	This
is	not
a	pipe

may be atomized to:

**atoms	**atoms
1	2
3	4
5	6
6	1
2	3
4	7

Internally, it is these integers that are used when computing the Damerau-Levenshtein edit-distance.

Since *simil* converts all data tokens into integers, the only thing *simil* knows how to do with two data tokens is to tell whether or not they are identical. This highlights the fact that *simil* does not *interpret* the text of any data tokens it reads. In using the *simil* program, the *qualitative* similarity decision is made by the choice of interpreted humdrum input. For example, the pitch similarity of two strings can be measured by providing absolute pitch data as input. Alternatively, the melodic contour similarity between two strings can be measured by providing input representing pitch contour, and so forth. Since humdrum permits the user to define arbitrary representation types, the user is free to define any qualitative dimension that may be of interest. For example, this scheme would permit a user to measure the similarity of vowel content between the Russian original and French translation of lyrics in a choral work. In this case the appropriate input given to *simil* would represent vowel phonemes.

The Operation of *Simil*

The *simil* program accepts two humdrum files as input. The two files are referred to as the *source file* and *target file*. In addition to the input, the output of *simil* also conforms to the humdrum syntax -- outputting one or two columns of data depending on the mode of operation. The first column always outputs the similarity measures, whereas the second column, when present, provides information identifying the location of highly similar strings. Similarity measures are continuous numerical values ranging between zero and one. These numbers indicate the relative similarity of the data in the source input to data in the target

input. In order to *normalize*, or "force" this number into this range, the Damerau-Levenshtein edit distance is scaled using the expression where *d* is the Damerau-Levenshtein distance, *l* is the length of the template measured in tokens, and *e* is Euler's constant. [4] [5] This means that an output value near zero indicates great dissimilarity of the template and the primary input at the current location, whereas a number near one indicates a high level of similarity. A value of exactly one indicates that the Damerau-Levenshtein distance is zero, and so the source and target inputs match exactly at the current location. Note that a value of exactly zero cannot be achieved in theory, although due to roundoff the value 0.00 could appear in the output.

The *simil* command provides two modes of operation: a *fixed template mode* and a *moving template mode*. In the fixed template mode the user specifies a (typically short) sequence of data tokens in the source input that defines a pattern "template." The target input is then searched for features that are similar to the template. The program produces a single humdrum output spine labelled **simil. Once again, the output values range from zero to one according to a Damerau-Levenshtein metric. The edit-distance is measured between the entire template and a string of corresponding length beginning at the current point in the target input. Successive output values indicate the degree of similarity between the target and source inputs as the template is shifted along the target input. Note that if the target and source inputs are the same length, only a single similarity measure is produced. If the source input is *n* tokens shorter than the target input, then *n* similarity values are output. Note that in fixed template mode, the template is taken to be the entire source input. In moving template mode, by contrast, the source input is broken up to provide many individual templates. The user specifies some fixed template *length* as a command line option. The source input is typically much longer than this length. Each set of <length> consecutive tokens in the course input is used as a pattern template. For example, if the template is defined as five tokens in length and the source input contains twenty tokens, then a total of sixteen individual templates will be scanned -- consisting of tokens 1-2-3-4-5, 2-3-4-5-6, 3-4-5-6-7, ... 16-17-18-19-20.

As each data token is encountered in the target input, *simil* scans all possible templates in the source input in order to determine which template exhibits the lowest Damerau-Levenshtein edit-distance. This minimum value is output along with the line number in the source input from which the minimum distance was calculated. This allows the user to identify the location of the similar features in the source input. [6] It is possible that more than one template produces the same minimum edit-distance for a given point in the target input. All such templates are identified by *simil*. If a user knows exactly what feature is being sought, then the user would typically select the *fixed* template mode. The *moving* template mode might be used as an automated way of discovering similar features without the need for user intervention or guidance. A typical example of the use of the two modes might be to use the moving template mode to find some interesting similarities, and then extract one of the similar features to be

used as a fixed template.

Weighting the Edit Distance

In the foregoing discussion, we have assumed that, in computing the similarity between two strings, the cost of each edit operation is +1. *Simil* allows the user to define the cost of each edit operation via an initialization file. Arbitrary costs may be assigned to any of eight edit operations (shown in Table 1). In describing the edit operations, String 1 is the source string and String 2 is the target string. Notice that there is no overt edit operation for insertion: an insertion in String 1 is deemed equivalent to a deletion in String 2. However, different edit penalties may be defined for deletions from String 1 (D1) compared with deletions from String 2 (D2). In musical applications defining such asymmetrical penalties may be important. For example, two inputs may represent a basic melody and an embellished variant of the melody. We may wish to define the deletion of tones from the embellished version as less costly than deletion of tones from the basic melody.

Table 1: Basic Edit Operations in *simil*

Name Tag	Edit Operation
D1	Delete a nonrepeated token in String 1
D2	Delete a nonrepeated token in String 2
R1	Delete a repeated token in String 1
R2	Delete a repeated token in String 2
S0	Substitute a token that is repeated in neither String 1 nor String 2
S1	Substitute a token that is repeated in String 1 only
S2	Substitute a token that is repeated in String 2 only
S3	Substitute a token that is repeated in String 1 and String 2

Since repetition is a common form of musical variation, *simil* allows the user to distinguish between repeated and non-repeated tokens. A repeated token is defined as one that is immediately preceded by an identical token. Thus, in deleting a sequence of identical symbols in String 1, say, all deletions except the first occurrence are R1 operations, whereas the deletion of the first occurrence is a D1 operation.

Note that the minimum theoretical edit-distance for any set of penalty weightings can be determined empirically by providing the *simil* program with source and target strings that share no symbols in common. For example, the source input may consist entirely of numbers, whereas the target input consists entirely of alphabetic characters. In the case where all edit operations are assigned a penalty of +1, the minimum quantitative similarity between two strings is 0.37.

Some user-defined weightings may give rise to peculiar results -- such as negative costs -- but *simil* does not forbid this. *Simil* does offer warning messages if the weighting seems unduly strange; for example, if the cost of R1 is more than that of D1. In addition, *simil* will abort operation if the defined edit penalties transgress the triangular inequality. The default weighting for all operations is +1. Ideally, we would prefer to be able to establish appropriate edit penalties by consulting data arising from perceptual experiments on the perceived relative similarity of various types of musical manipulations. However, in the absence of such data, analysts may want to rely on intuition in assigning the costs associated with various edit operations. The fact that relevant perceptual data is not currently available does not invalidate *per se* an intuitive approach. It only means that the resulting quantitative measures of similarity may be rather crude.

Applications to Musical Problems

The benefits and limitations of using the edit-distance to measure similarity can be illustrated by considering some applications of the *simil* program to musical problems. By way of illustration, we will consider three applications. In the first application we will use *simil* to find instances of similar material within a single work. In the second application we will examine the effect of two different representations of rhythm on measures of rhythmic similarity. In the third application we will compare the relative similarity of three musical works using a large number of qualitative views of musical similarity.

Two-part Invention No. 1 by J.S. Bach

Consider first, the simple task of locating motivic statements in a musical work. Pattern grammars are normally successful in identifying a range of musically useful patterns. However, pattern grammars often fail to identify trivial variants of the sought pattern. The complexity of the task is compounded by the fact that users usually have insufficient experience to know how to define a suitably embracing pattern template. Using a similarity approach to this problem provides a more "forgiving" approach to pattern-matching tasks. Figure 2 shows the opening measures of well-known keyboard invention No. 1 by J.S. Bach (Steglich, 1979). The first eight notes of the opening motive have been circled. Suppose we wanted to locate instances of the motive throughout the work. Choosing a pitch-specific representation would make it impossible to identify transpositions of the motive. A more appropriate representation is a melodic interval representation where each token represents the direction (up/down) and degree (in semitones) of pitch change from the previous note.

Figure 2. Opening measures of J.S. Bach, two-part Invention No. 1 for clavier. The first eight notes of the opening motive are circled.

The upper and lower voices can be extracted and used as separate target inputs. *Simil* outputs a column of numbers indicating the relative similarity of the two input files at each point in the voice. Figures 3a and 3b plot the instantaneous similarity for the treble and bass voices respectively.

Figure 3a-b: Instantaneous similarity between the opening eight-note motive and the complete treble (**3a**) and bass (**3b**) voices for Bach Invention No. 1. The X-axis represents the score position (in measures). The Y-axis plots the similarity of the current position to the opening eight-note motive shown in figure 2. Exact quotes of the motive (including real transpositions) are indicated by similarity values of 1.0

The highest peaks correspond to exact statements of the motive -- points that could be identified using any exact pattern match approach. Lesser peaks (ranging from 0.75 to 0.87) correspond to near-exact quotations of the motive. Of the similarity instances detected, the last instance in the bass voice is probably the most instructive. Figure 4 shows the last two measures of the piece with a similarity instance circled in the lower voice. Bach uses augmentation during the first half of the measure -- doubling the note values from sixteenths to eighths -- whereas the second half of the measure continues the motive with the original rhythm, but with changes of pitch. Most theorists would have little difficulty recognizing the similarity between this passage and the opening motive. However, the similarity is sufficiently subtle that a simple pattern matching program would fail to identify it.

Figure 4. Final measures of Bach's two-part Invention No. 1. The circled passage in the bass voice was determined to have a similarity value of 0.75 to the opening motive. Such instances are difficult to detect using conventional pattern-matching techniques.

A Rhythmic Example

The term "rhythm" is a catch-all phrase that tends to obscure a number of ways of understanding the temporal organization of musical events. One way of characterizing a rhythm is as a list of *time spans*. According to this interpretation, each signifier represents a span of elapsed time between successive events. This conception is implied when we speak of a rhythm as "a dotted-quarter followed by an eighth and four sixteenths." Using a reciprocal duration notation, we might construct a corresponding humdrum input as follows:

**recip
4.
8
16
16
16
16

In representing time spans we normally distinguish between rests and notes, thus "a dotted quarter" is not the same as "an eighth note followed by a quarter rest."

Another way of characterizing a rhythm is as a list of *onset moments*. In this conception, we are interested simply in marking the onsets of notes -- without regard to the note's duration or the occurrence of rests. One way of representing this is where successive data records represent equivalent elapsed durations and onsets are marked by the presence or absence of marker tokens. In the following humdrum representation, for example, each successive data record (line) represents a span of an eighth note. The marker tokens (X) indicate onsets consistent with two successive quater-notes, followed by two eighth-notes, followed by a half-note:

**timebase
X
.
X
.
X
X
X
.
.
.

Notice that the same representation would be used for an eighth-note, followed by an eighth rest, followed by a quarter-note, followed by three eighth notes, followed by a dotted-quarter rest. That is, the onset structure would be identical -- although the duration structures are entirely different.

Neither the time-span representation nor the onset-moment representation given above link the rhythms to any metrical context. Since meter is an important aspect of rhythmic perception, yet other forms of rhythmic representation may be distinguished by anchoring events to their respective metric positions.

Assume, for example, that the above time-base rhythm began at the beginning of a measure of 2/4 meter. We could use numerical values to represent the position of successive events in the metric hierarchy (1=downbeat at the beginning of the measure; 2=second most prominent metric position; 3=third most prominent metric position, etc.)

**metricposition
1
2
1
3
2

A time-span or timebase notation could be augmented by metric position information in order to provide some composite rhythmic representation. Note, once again, that when we represent a rhythm, our choice of representation implies a set of qualitative assumptions regarding the perceptual or analytic importance of a given way of encoding rhythm. The importance of these qualitative assumptions in measuring rhythmic similarity are illustrated by the three rhythms notated in Figure 5. These three rhythms were each represented using two different humdrum interpretations: *timebase* and *duration*. (For simplicity we will avoid metric context information here.)

Figure 5. Three simple rhythms. The first and second rhythms are most similar with respect to durational organization. The first and third rhythms are most similar with respect to onset-moment or timebase organizations. See Tables 2a and 2b.

Notice that rhythms notated in the timebase representation are easily shifted; in particular, a syncopated rhythm that begins at a weak metric position can be readily transformed into one that begins at a strong metric position (and vice versa), simply by inserting or deleting the first few tokens. In Figure 5, notice that Rhythm 2 can be made very similar to Rhythm 1 by deleting the initial eighth-note rest in Rhythm 2. From Tables 2a & 2b it can be seen that using the timebase representation, Rhythms 1 and 3 are most alike, whereas using the duration representation, Rhythms 1 and 2 are most alike. This example highlights the importance of identifying the qualitative framework within which similarity measures are made. In the next sample problem, we will examine more than a dozen qualitative dimensions in characterizing musical similarity.

Table 2a

Similarity of Three Rhythms: Timebase Representation

	Rhythm 1	Rhythm 2	Rhythm 3
Rhythm 1	1.00	0.78	0.88
Rhythm 2		1.00	0.73
Rhythm 3			1.00

Table 2b

Similarity of Three Rhythms: Duration Representation

	Rhythm 1	Rhythm 2	Rhythm 3
Rhythm 1	1.00	0.72	0.61
Rhythm 2		1.00	0.43
Rhythm 3			1.00

Three Chorales by J.S. Bach

Throughout his life, Bach arranged innumerable hymn melodies popular in his day. In a few cases, Bach harmonized the same melody on different occasions. One such chorale is "Lobt Gott, ihr Christen allzugleich" which is harmonized as chorales No. 54 and No. 136 in the collection assembled by Riemenschneider (1941). The two harmonizations are shown in Figure 6 along with a third unrelated chorale harmonization (No. 136). Notice that the two harmonizations of "Lobt Gott ..." are quite alike, but different in a number of ways. The soprano line (melody) in both harmonizations is nearly identical -- with a few notable differences. In the second phrase (measure three) the melody lines differ (E5 -> E5 rather than D5 -> C#5). In harmonization No. 54 a rest has bee notated in the fourth measure that is not present in No. 276. In beat three of measure five of No. 276 an unaccented passing tone has been added (C5). In the subsequent measure, the rhythm has been changed slightly (*eighth -> eighth -> quarter*) rather than (*quarter -> eighth -> eighth*). Finally, another change occurs in the melody line in the middle of the penultimate measure (m.9).

Chorale No. 54

Figure 6. Three Chorale harmonizations by J.S. Bach. Chorales nos. 54 and 276 are different renditions of "Lobt Gott, ihr Christen, allzugleich." Chorale no. 136 is unrelated, "Herr Jesu Christ, dich zu uns wend."

Chorale No. 276

Figure 6b. Second rendition of "Lobt Gott, ihr Christen, allzugleich."

Chorale No. 136

Figure 6c. A chorale in the same key, but otherwise unrelated to Chorales 54 and 276. "Herr Jesu Christ, dich zu uns wend."

The primary differences between chorales No. 54 and No. 276 are to found in the choice of harmonies and in the part-writing. In some places (such as the first two measures) the harmonies are very similar, whereas in other places (notably measures eight and nine) the harmonies differ markedly. For demonstration purposes, we have selected a third chorale (No. 136) as a "foil" against which we can compare various similarity measures. By running *simil* on several alternate representations of the musical data, we can explore a finite set of qualitative dimensions of similarity. By way of illustration, we will examine nine qualitative approaches to the part-writing as well as several additional qualitative approaches to the harmony. For each qualitative approach we must represent the musical passage using a different humdrum representation.

Tables 3a-c compare the similarities for each of the pairings of the three sample Bach chorales. Each table shows the results of running *simil* for all of the four parts: soprano, alto, tenor, and bass. For example, the values in the first row of Table 3a arise by comparing the soprano part in chorale No. 54 with the soprano part in chorale No. 276. Each column in Tables 3a-c indicates the similarity measure for a different qualitative dimension. Each note in a given part is compared with corresponding notes in the other two chorales. The first numerical column identifies the similarity for *pitch+duration* information. In this case, notes are deemed to be "identical" only if they have both the same notated pitch and duration. The second and third data columns pertain to duration only without regard to pitch information. For these measures, notes are deemed "identical" if they share the same notated duration. Bach was in the habit of writing pauses above notes ending each phrase. The second data column includes the pause data within the representation of duration, whereas the third data column excludes the pause -- and so is insensitive to implied phrase information. As we saw earlier, there are different ways of understanding "rhythm." Consequently, we have distinguished "durational" representations (columns two and three) from "timebase" representations (column four). In these measures, only the rhythmic information has been preserved; all pitch-related information has been ignored.

Table 3a

Similarity Measures for J.S. Bach Chorale Nos. 54 and 276

	pit+dur	durat§	durat§§	timebase	pitch¶	diatonic	semits	pc	contour°	contou
sopr.	0.72	0.83	0.83	0.94	0.91	0.91	0.91	0.91	0.87	0.89
alto	0.51	0.64	0.69	0.86	0.65	0.65	0.65	0.65	0.62	0.66
tenor	0.48	0.65	0.66	0.87	0.57	0.61	0.57	0.59	0.62	0.70

bass	0.41	0.62	0.65	0.85	0.54	0.63	0.54	0.61	0.59	0.78

Legend:

pit+dur	notated pitch plus duration information (including pause markings)
durat§	duration information (including pause markings)
durat§§	duration information (excluding pause markings)
timebase	time-base information
pitch¶	notated pitch (G# is not equal to A-flat; octave differentiated; rests eliminated, tied-notes treated as a single pitch)
diatonic	diatonic (letter-name) pitch information
semits	pitch height in semitones (rests eliminated, tied-notes treated as a single pitch)
pc	pitch-class information
contour°	contour (5 motions: up-step, down-step, up-leap, down-leap, same) based on semitone information
contour¤	contour (3 motions: up, down, same) based on semitone information

As we have done for duration, it is also possible to isolate just the pitch information for each note. Pitch can be regarded from several qualitative points-of-view. Data columns five through seven identify similarity measures for three different interpretations of pitch. In each case, rests have been eliminated and tied-notes have been treated as a single pitch. Column five distinguishes each pitch according to how it is *notated*. Specifically, octave information is distinguished (A4 is not equal to A5) as well as enharmonic spelling (B-sharp is not equal to C). In column six, pitches are distinguished according to just the diatonic "note-name". In this case, only seven categories are distinguished corresponding to the diatonic pitches (A-G). Since the chorales share the same key, a diatonic pitch representation is equivalent to a scale degree representation (1-7). Column seven expresses pitches in terms of semitone pitch height. In this interpretation, enharmonic spellings are deemed to represent the same pitch (B# = C). Column eight reduces the pitch data further by expressing pitches in terms of pitch-classes: both octave information and enharmonic spelling are irrelevant. The final two columns in Tables 3a-c pertain to measures of contour similarity. In column nine, contours are distinguished by five categories of motion: (1) up-step, (2) down-step, (3) up-leap, (4) down-leap, and (5) same pitch. In column ten, contour information has been reduced to just three categories: (1) up, (2) down, and (3) same pitch. In both cases, the contour calculations have been based on pitch-height information. Depending on the analytical perspective, a user might want to distinguish a different contour measurement in which contour was calculated from diatonic scale degree information (e.g., *B* may be regarded as

having the same diatonic height as *B-sharp*, but *B-sharp* might be regard as being "lower" in diatonic height than *C*).

Table 3b

Similarity Measures for J.S. Bach Chorale Nos. 54 and 136

	pit+dur	durat§	durat§§	timebase	pitch¶	diatonic	semits	pc	contour°	contou
sopr.	0.43	0.70	0.75	0.90	0.54	0.56	0.54	0.54	0.60	0.65
alto	0.43	0.70	0.77	0.87	0.56	0.56	0.56	0.56	0.58	0.63
tenor	0.44	0.67	0.76	0.90	0.52	0.52	0.52	0.52	0.51	0.64
bass	0.42	0.62	0.65	0.82	0.52	0.58	0.52	0.56	0.62	0.74

Legend:

pit+dur	notated pitch plus duration information (including pause markings)
durat§	duration information (including pause markings)
durat§§	duration information (excluding pause markings)
timebase	time-base information
pitch¶	notated pitch (G# is not equal to A-flat; octave differentiated; rests eliminated, tied-notes treated as a single pitch)
diatonic	diatonic (letter-name) pitch information
semits	pitch height in semitones (rests eliminated, tied-notes treated as a single pitch)
pc	pitch-class information
contour°	contour (5 motions: up-step, down-step, up-leap, down-leap, same) based on semitone information
contour¤	contour (3 motions: up, down, same) based on semitone information

Table 3c

Similarity Measures for J.S. Bach Chorale Nos. 136 and 276

	pit+dur	durat§	durat§§	timebase	pitch¶	diatonic	semits	pc	contour°	contou
sopr.	0.45	0.68	0.72	0.87	0.54	0.56	0.54	0.54	0.60	0.65
alto	0.42	0.66	0.68	0.86	0.59	0.61	0.59	0.59	0.62	0.65
tenor	0.43	0.61	0.65	0.79	0.54	0.55	0.54	0.55	0.54	0.66
bass	0.39	0.65	0.68	0.79	0.46	0.55	0.46	0.51	0.58	0.71

Legend:

pit+dur	notated pitch plus duration information (including pause markings)
durat§	duration information (including pause markings)

durat§§	duration information (excluding pause markings)
timebase	time-base information
pitch¶	notated pitch (G# is not equal to A-flat; octave differentiated; rests eliminated, tied-notes treated as a single pitch)
diatonic	diatonic (letter-name) pitch information
semits	pitch height in semitones (rests eliminated, tied-notes treated as a single pitch)
pc	pitch-class information
contour°	contour (5 motions: up-step, down-step, up-leap, down-leap, same) based on semitone information
contour¤	contour (3 motions: up, down, same) based on semitone information

Having described each qualitative dimension shown in Tables 3a-c, we can make a few observations about the findings. With regard to chorales Nos. 54 and 267 (Table 3a), the most obvious result is that the soprano parts are most similar of all the parts. No matter what qualitative property we examine, the similarity values for the soprano contains strictly less information. That is, notated pitch contains more information than enharmonic pitch-height, which in turn contains more information than pitch-class representation. As the amount of information is reduced, similarity values can be expected to increase by chance alone. Since this increase has not happened, we can conclude that of the various pitch representations examined, the two soprano lines are most similar with respect to notated pitch. The high similarity values regarding enharmonic pitch-height and pitch-class are simply consequences of the close similarity of the notated pitches.

The situation is somewhat different in the case of bass lines. With regard to chorales Nos. 54 and 267, the *pitch-class* similarity is somewhat higher than the *notated-pitch* and *pitch-height* representations. This means that the bass lines often share the same pitch-class (say *G*) but differ in pitch-height (*G2* versus *G3*). This situation can be contrasted with the alto parts, which show no improvement in similarity measures in moving from notated-pitch to pitch-height to pitch-class representations. The range of the alto voice is much more constrained than the range of the bass voice -- with the result that there is much less octave displacement of shared pitch-classes.

Tables 3b and 3c provide comparable similarity measures between each version of "Lobt Gott, ihr Christen, allzugleich" and the unrelated chorale "Herr Jesu Christ, dich zu uns wend" (No. 136). "Herr Jesu ..." is two measures shorter than either chorales Nos. 54 and 276. Thus, many more deletion edits must take place in order for this chorale to be matched with the other two. Comparing the values in Tables 3b & 3c with the comparable values in Table 3a, we note that thirty (of thirty-eight non-equal values) are higher than the corresponding values in Table 3b. Similarly,

thirty-four (of thiry-eight non-equal values) are higher in Table 3a than the corresponding values in Table 3c. These differences are significant (chi-square = 12.7; df=1; *p*=0.000359) and (chi-square = 23.7; df=1; *p*=0.000001). These numbers affirm the observation that chorales No. 54 and No. 276 are more like each other than either is like chorale No. 136.

In comparing Tables 3b and 3c, we find that nineteen (of thirty-four non-equal values) are higher in Table 3b than the corresponding values in Table 3c. These differences are not significant (chi-square = 0.47; df=1; *p*=0.49), so we cannot say that chorale No. 54 is more like chorale No. 136 than is chorale No. 276 like chorale No. 136. More precisley, given the various qualitative musical dimensions examined, we cannot say that either No. 54 or No. 276 is more like chorale No. 136.

Harmonic Similarity

Table 4 provides yet another set of similarity measures related to harmony. For each of the three chorales, traditional functional harmonies were identified using "roman numeral"-type analysis. These were then encoded in various ways as humdrum data. The three right-most columns in Table 4 identify all possible pairings of the chorales: Nos. 54 & 276, Nos. 54 & 136, and Nos. 136 & 276. The left-most columns indicate the various representational distinctions. For example, in some cases different chord inversions were distinguished (e.g., *I6* differs from *I*). In other cases, no distinction was made between those chords with or without tertian extensions (e.g., *V7=V*). In other cases, all chords having the same scale degree as root were deemed equivalent (e.g., *ii=II=V/V*). In addition, some representations marked the cadence points -- distinguishing an incidental *V-I* progression from a *V-I* authentic cadence occurring at the end of a phrase. Finally, a rest occurs in choral No. 54. In some comparisons the rest was omitted, whereas in other comparisons the presence of the rest was treated as a harmonically pertinent symbol.

Table 4.

Harmonic Similarity Measures for Chorales Nos. 54, 136, and 276.

Inversions	7ths	Degree	Cadence	Rest	Nos.54/276	Nos.54/136	Nos.136/276
§	§	§	§	§	0.53	0.46	0.49
§	§	§	§	x	0.54	0.44	0.49
§	§	§	x	x	0.54	0.48	0.51
x	§	§	§	§	0.57	0.51	0.49
x	§	§	§	x	0.59	0.51	0.49
x	§	§	x	x	0.61	0.55	0.56
§	x	§	§	§	0.56	0.47	0.49
§	x	§	§	x	0.56	0.46	0.49
§	x	§	x	x	0.56	0.50	0.51

x	x	§	§	§	0.61	0.52	0.53
x	x	§	§	x	0.61	0.52	0.53
x	x	§	x	x	0.62	0.57	0.59
§	§	x	§	§	0.53	0.46	0.49
§	§	x	§	x	0.54	0.44	0.49
§	§	x	x	x	0.54	0.48	0.51
§	x	x	§	§	0.57	0.48	0.49
§	x	x	§	x	0.57	0.48	0.49
§	x	x	x	x	0.57	0.52	0.51
x	§	x	§	§	0.57	0.51	0.51
x	§	x	§	x	0.59	0.51	0.51
x	§	x	x	x	0.61	0.55	0.59
x	x	x	§	§	0.62	0.55	0.57
x	x	x	§	x	0.62	0.55	0.57
x	x	x	x	x	0.64	0.60	0.66

Legend:

§	distinguished
x	not distinguished
inversions	root, 1st, 2nd inversions
7ths	triad, seventh chords
degree	ii, II, V/V
cadence	cadence chords
rest	musical rest

With the exception of a single similarity value, all values in Table 4 show chorales Nos. 54 and 276 as being most harmonically similar. Only the last value in Table 4 departs from this trend. The last row of values are based on a bare-bones harmonic representation in which no distinction is made between chord inversions, between triads and seventh chords, between various types of chords having the same scale degree as root, and where no distinction is made between chords at cadence points, and the presence of rests is ignored. This single value appears to show that with respect to this qualitative notion of harmonic similarity, chorale No. 54 is more like chorale No. 136.

Other Possible Applications

Clearly, there are many ways in which the Damerau-Levenshtein metric for edit distance could be applied to problems in music analysis. *Simil* could be used to

analyze a set of variations -- giving a quantitative measure in several dimensions at once over the entire set of variations. Thus it would be possible to characterize empirically the degree of similarity between different variations, to identify the qualitative dimension in which all variations are most similar to the theme, and to identify the qualtitative dimension by which successive variations are different -- that is, the precise meaning of variation. A similar approach could be used to characterize "the similarity" of all works of a given composer or stylistic period. Thus it may be possible to characterize the distinctiveness of a given composer or manner of composition. It might also be the case that such an approach would be helpful in cases of conflicting attribution. It bears emphasizing, however, that the numerical values arising from *simil* depend on the edit-distance penalties embodied in the defined metric. Considerable experimentation and experience is needed in order to establish and refine perceptually-appropriate similarity metrics. Without such information, *simil* will remain a somewhat intuitive tool.

A possible extension to the use of the Damerau-Levenshtein metric wold be to implement a recursive design. The *simil* program could be modified so that it outputs the edit sequences for the minimum-cost edit path between two music-related representations. These edit sequences could themselves be treated as representations of musical manipulations. Hence the output could be fed back to *simil* as input, and similarities could be computed between two sets of edit operations. This would permit the detection of "second-order" similarities. That is, we could address such questions as: "Is the way this material has been varied similar to the way this other material was varied?" This technique might prove to be a powerful method for characterizing and comparing variational techniques and compositional processes. By way of example, a common variation technique is to embellish a melody using triplet figurations. When comparing a variation with the original melody, the edit path will be highly patterned: a repetition of the basic pattern of skipping over one melody tone and deleting two embellishment tones. This pattern of edit operations would be very similar for quite different variations that employed triplet embellishment.

Conclusion

In this paper we have explored one approach to the quantitative measure of similarity between non-parametric data. We noted that the problem of similarity entails two aspects: the *qualitative* question of "in what way are two things similar?" and the *quantitative* question of "to what degree are two things similar?" In principle, any qualitative notion of similarity may be entertained. But in practice, appropriate qualitative notions will be constrained by how humans experience or conceive of a given phenomenon. In the case of the *simil* program, the qualitative conception of similarity is expressed concretely through the choice of musical primitives to be represented. These qualities may include (but are not limited to): absolute pitch, relative pitch, diatonic pitch, scale-degree, pitch-class, melodic interval, diatonic interval, semitone interval, pitch contour, diatonic

contour, duration, time-base onset structure, metric position, sonorities, chord spelling, functional harmony, dynamics, lyrics, and Schenkerian graphs. When provided with an appropriate input representation, *simil* is as adept at measuring the similarity of background structures or generative processes, as it is at measuring the similarity of foreground elements such as pitches.

When numerical data are available, it is possible to use various parametric methods to measure similarity, such as Pearson's coefficient of correlation. However, qualitative data resist such parametric approaches. One approach to characterizing the quantitative similarity for non-quantitative data is to use the Damerau-Levenshtein metric for edit distance. According to this approach, the degree of similarity may be determined by asking what changes must be made to one object in order to make it identical to another object. Suitable editing operations may include: deletion, insertion, dissimilar substitution, or repetition. Each operation may be assigned a numerical penalty expressing the degree of dissimilarity arising from such a change. The numerical penalties used are constrained by the traingular inequality criterion. In addition, penalties will depend on the field of application. In music, repetitions may have a relatively low penalty, whereas reversing the order of two elements may have a higher penalty. In the case of typing errors, by contrast, the penalties may be the opposite: repetitions may carry a relatively higher penalty, whereas reversing the order of two characters may have a lower penalty. Determining the most appropriate field-specific edit penalties will require extensive perceptual experimentation and experience.

Notwithstanding the assumptions and limitations of this approach, it appears that metrics based on the Damerau-Levenshtein edit-distance provide a promising way of characterizing the quantitative similarity between musical passages. [7] [8]

Endnotes

[1] All programming was done by the first author. The write-up was produced by the second author in conjunction with the first author. Return to text.

[2] In the case of Unix environments, for example, such patterns may be defined using regular expression syntax. Return to text.

[3] An explanation of Pearson's coefficient of correlation can be found in any introductory statistics textbook. Return to text.

[4] Euler's constant is e=2.7182818... There is nothing particularly deep about this choice; it is just the most mathematically natural base for exponentials. Return to text.

[5] Another way to compute similarity based on edit distance would be to use the

formula , where *d* is the edit distance as before. This has the virtue of being simpler, but is has some anomalous repercussions. When we compute the similarity using the formula: , the dependence of the similarity values on -- rather than on *d* alone -- means that the total similarity depends on the *ration* between the edit distance and the length of the template. This scaling ensures that the use of longer templates are not unduly penalized due to the large edit distances one would expect for long templates. Suppose, for some given template, that the minimum number of edits needed to make it match a given source string was one tenth of the length of the template. We would like the two strings to produce the same similarity value, regardless of their length. If the length of the template was 10 (say) then one edit operation would be required, and so the similarity by the formula would be , or about 0.37. With a template length of 100, however, the similarity drops dramatically to , or 0.000045. In this situation, computations of similarity values using different template lengths do not give comparable results. If the formula is used, however, both calculations give the same result. This is not to say that the formula is *correct* in any formal sense -- merely that it provides a practical way of scaling data for strings of different length. For most applications, users won't be interested in comparing similarity values calculated using different template lengths. Return to text.

[6] In the implementation, this cross-reference information is output in the "**simxrf" spine. Cross-references are provided only for those places in the source input which correspond to the minimum edit-distance, and hence some features that are very nearly as similar could be ignored. Return to text.

[7] Copies of the *simil* program described in this paper may be obtained free of charge by writing to David Huron, Music Department, Conrad Grebel College, University of Waterloo, Waterloo, Ontario, Canada N2L 3G6. Return to text.

[8] Software support by Mortice Kern Systems is gratefully acknowledged. Return to text.

References

Aho, A.V. & Johnson, S.C. (1974). LR Parsing. *ACM Computing Surveys* Vol. 6, No. 2, pp.99-124.

Bach, Johann Sebastian (1979). *Inventionen und Sinfonien* (BWV 772-801). R. Steglich, (ed.) München: G. Henle Verlag.

Damerau, F.J. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM*. Vol. 7, No. 3, pp. 171-176.

Hall, P.A.V., & Dowling, G.R. (1980). Approximate string matching. *ACM Computing Surveys*, Vol. 12, No. 4, pp. 381-402; reprinted in S.N. Srihari (eds.) (1985). *Computer Text Recognition and Error Correction*. Silver Spring, MD: IEEE

Computer Society Press, pp. 17-38.

Huron, D. (1988). Error categories, detection and reduction in a musical database. *Computers and the Humanities*, Vol. 22, No. 4, pp. 253-264.

Huron, D. (1991). *The Humdrum Toolkit: Reference Manual*. Menlo Park, California: Center for Computer Assisted Research in the Humanities, 552 pages, ISBN 0-936943-10-6.

Huron, D. (1992). Design principles in computer-bsaed music representation. In A. Marsden & A. Pople (eds.), *Computer Representations and Models in Music*. London: Academic Press, pp. 5-39. Abstract

Kernighan, B.W., & Ritchie, D.M. (1988). *The C Programming Language*. (2nd edition). Englewood Cliffs, NJ: Prentice-Hall.

Levenshtein, V.I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10, pp. 707-710.

Riemenschneider, A. (1941). *371 Harmonized Chorales and 69 Chorale Melodies with Figured Bass by Johann Sebastian Bach*. New York: G. Schirmer.

Ullman, J.R. (1977). A binary n-gram technique for automatic correction of substitution, deletion, insertion and reversal errors in words. *Computer Journal*, Vol. 20, No. 2, pp. 141-147.

Wagner, R.A. (1972). An n^3 minimum edit-distance correction algorithm for context-free langauges. Nashville, Tenn.: Technical Report, Systems and Information Sciences Department, Vanderbilt University.

David Huron's Home Page
List of Publications by David Huron