CHAPTER NINE

The Evolution of Musicat

In the introduction to this dissertation I wrote that Musicat is a model of music listening. But it is misleading to call it simply *a* model, in the singular, because it is actually a work in progress and has evolved significantly from its original form. The original versions of Musicat shared the same underlying FARG architecture, but share surprisingly little code with the current program. Most significantly, I rewrote the bulk of the program starting in September 2011, and it reached its current state in the early summer of 2012. The listening performances of Chapters 5–7 and the description of the program's architecture in Chapter 8 are based on that latest version. This chapter, however, covers the previous versions of Musicat. Many of the ideas from previous versions did not make it into the latest version, but I think some of them are important and should be resurrected in future incarnations of Musicat.

In addition to showing old ideas in prior versions of Musicat, this chapter also highlights some of the problems that plagued Musicat in the past. To be sure, it has plenty of problems in its latest version, but early versions had even more issues, and a discussion of those problems may prove even more illuminating than any of Musicat's successes.

A Brief History of Musicat

Because Musicat has been evolving gradually over time as I have tried out new ideas and focused my energy on different aspects of the program, it is difficult to clearly delineate different versions of the program. However, I can describe several time periods of development and mention the most salient ideas and issues of those times. In addition, there were a few major architectural changes that took place, and those turning points are key moments that invite the use of different names for the program. Some of the most significant time periods and architecture changes are listed below:

- Spring 2004–Spring 2008: Initial work. Hierarchical, Schenkerian, based on Larson's multilevel Seek Well model. I call the program from this time "**MusicatH**", for "Musicat with Hierarchy".
- Fall 2008–Fall 2009: Removal of hierarchy (starting here, the program is named "**MusicatW**", for "Musicat without Hierarchy"). Addition of more complicated rhythms. Development of a detailed user interface for adjusting parameters. Attempts at machine learning of parameters. Addition of "perspects" to notes and groups.
- Spring 2010: Semester in Paris. Increased use of motivic memory, and spreading activation to similar motifs. Improvements to user interface, including visualizations of codelet distribution on the Coderack. Some effort to handle larger-scale melodies such as "Mademoiselle de Paris" (previously, only 4-measure long melodies had been used extensively). Groups are forced to start and end at bar lines instead of mid-measure.

- Fall 2010–Spring 2011: More emphasis on the importance of chunks of music with powers-of-2 lengths. A new analogy data structure was added.
- Fall 2011: A new emphasis on the primacy of rhythmic structure in musical listening. Also, previous unresolved problems led me to try two radical new ideas:
    - **BinaryCat**: I wanted to focus on grouping and analogy-making at the measure-level and higher, so I toyed with the idea of representing each measure with a high-dimensional binary vector, inspired by *Sparse Distributed Memory* (Kanerva, 1988). This was just a fleeting experiment, but it led to:
    - **RhythmCat**: This was a complete rewrite of Musicat, intended to focus only on rhythm. Melodies were replaced by purely rhythmic patterns; notes no longer had pitches. Grouping and analogy-making were dependent exclusively on rhythm. Surprisingly, this experiment led to the current version of the Musicat.
- Spring 2012: RhythmCat evolved into the latest version of the program, simply called **Musicat**. I added pitch information back into RhythmCat, resulting in a program that "listens" to pitched melodies but generally treats rhythm as the primary musical feature. "Plug in" for Visual Studio was used to give me faster feedback about the program's behavior as I was making architectural experiments.

Versions of Musicat

This section describes the previous versions of Musicat at significant stages of its development. Three major versions are described: "Musicat With Hierarchy" ("MusicatH"), "Musicat Without Hierarchy" ("MusicatW"), and "RhythmCat". Afterwards, I make a few comments about how RhythmCat was converted into the latest version of Musicat. But first, a few comments on my early work on modeling the listening that happens at the very beginning of a melody.

Two-note Model (Key and Meter Induction)

My very earliest work on Musicat had a very different focus from all subsequent versions: the program was based on the FARG architecture, and it simulated melody perception but only for extremely short melodies. Specifically, my first goal was to make a program that was able to infer (or at least guess at) the key and meter of a melody very rapidly, after hearing just the first few notes. Steve Larson suggested this project to me in 2004. Reducing the size of the melody under consideration even further, he gave me the preliminary "assignment" of writing a program to implement what he called the "Two-note Model". Given just the first two notes of a melody, the Two-note Model would form a tentative idea about the key and meter of the melody. Then, additional notes would be given one at a time and the program would change its initial idea of the key and meter if necessary. For example, given the "Triplet Melody" discussed in Chapter 3, the desired program would initially hear the melody in 4/4 time in C major, but after the fourth or fifth note, it would change its interpretation to 3/4 time (still in C major). This model was related to Larson's earlier work in which he asked music students to sing continuations of two-note-long melodic fragments (Larson, 2004).

I started development on this model, which eventually led to the program "Musicat Without Hierarchy", discussed below. In parallel, I also ran a pilot study with human subjects (see Appendix A) to learn more about how people infer key and meter at the very start of a melody. After spending a significant amount of time on such issues, I came to realize (thanks to discussions with Douglas Hofstadter) that the activity of inferring key and meter was not at all central to our vision for the Musicat program, even though I think the FARG architecture is very well suited to solving this modeling problem. Instead of having Musicat spend time focusing on just two or three notes at the start of the melody, we wanted it to make musical analogies of various sizes and to listen to entire melodies of 16, 32, or possibly many more measures. In 2008 I stopped working on the key- and meter- inference part of the program and turned to larger-scale issues. Throughout Musicat's development, this trend continued: I made significant progress by focusing my attention (and the program's attention) on ever larger structures, as well as by devoting ever more attention to the idea of musical analogy-making.

MUSICAT WITH HIERARCHY (MUSICATH)

The first version of Musicat, counterintuitively, was in many ways the most sophisticated. Although its origin was in the Two-note Model program described above, I designed it on the basis of several discussions with Steve Larson about how his Multilevel Seek Well program could be converted into a program based on the FARG architecture. Musicat with Hierarchy (henceforth "MusicatH") was focused on generating melodic expectations for the next note (or the next several notes), using a Schenker-inspired multi-level analysis as well as small-scale note groupings.

Main Features of This Version

In order to make MusicatH more concrete and to demonstrate the main features of the program, I present a sample screenshot from June 2008 (I removed the explicit levels of hierarchy from the program shortly thereafter), using the melody in the following figure:

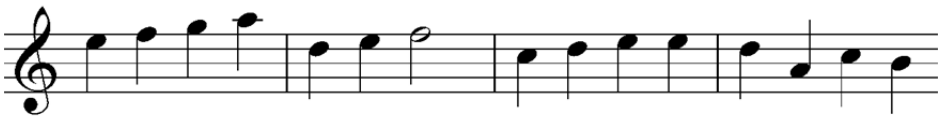

Figure 9.1: Opening of "Stückchen", from Schumann's *Kinderszenen*, Op. 15.

The melody of this Schumann piano piece was not a valid input for an early version of MusicatH, because only quarter notes were allowed, and this melody has a half note in measure 2. The program initially could "listen" to melodies from the Seek Well microdomain only (although I later added support for half notes, whole notes, and some dotted notes). Accordingly, I modified Schumann's melody by breaking the half note into two quarter notes (rests were not allowed; otherwise, I might have used a quarter rest on beat 4 of measure 2).

The next figure illustrates the kind of output that I had hoped this version of the program would generate. The figure shows the program during a run on the modified melody, but note that this is partly a sketch: the screenshot was modified slightly to make a *mocked-up* image of the desired user interface. The *groups* in the image were created by MusicatH, but the group *links* and "active motives" were added by hand).

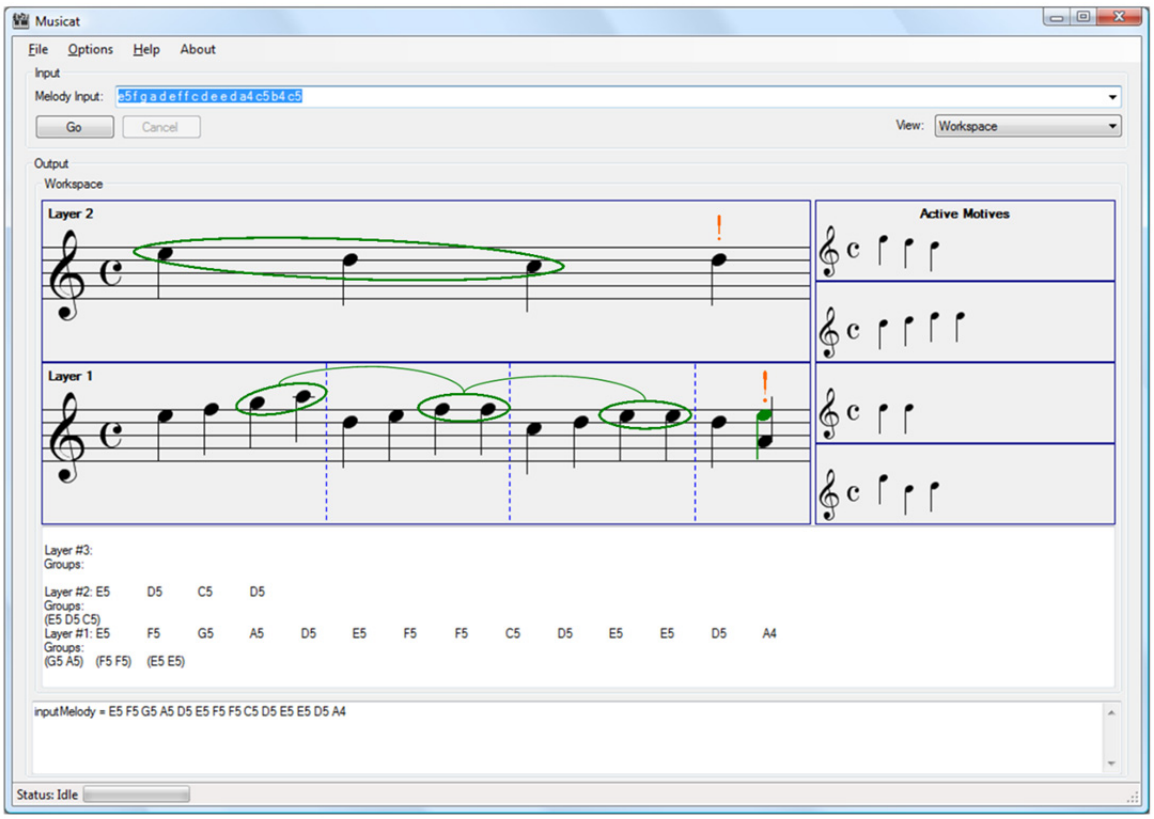

Figure 9.2: MusicatH listening to a modified version of "Stückchen".

Layer 1 in the figure contains the melody that the program is listening to. Layer 2 is generated by the program as it runs, and it consists of notes that have been "promoted" from Layer 1 because of perceived structural importance. The program theoretically had the ability to generate additional layers at even higher levels, but there were enough problems even in the two-layer case that I usually restricted the number of layers to two.

The green ellipses represent groups, just as variously-colored ellipses do in the current version of Musicat. MusicatH can make groups in any layer, and they consist of collections of adjacent *notes*, rather than of adjacent *measures*. There is no restriction for groups to start and end at measure boundaries. There are no meta-groups in this version of Musicat, but groups can overlap temporally with groups in a different layer (*e.g.*, in the figure, the first two groups in Layer 1 overlap in time with the group in Layer 2). Groups in the same layer cannot in

general overlap, although yellow groups, indicating pitch repetition, are an exception: a green group can overlap with a yellow group.

MusicatH is especially focused on noticing linear patterns and exact repetitions of pitch. For example, the groups formed in the figure are all linear motions, such as the **(E–D–C)** group in Layer 2 and the **(G–A)** group in measure 1 of Layer 1, or pitch repetitions, such as the **(F–F)** and **(E–E)** groups in Layer 1, measures 2 and 3.

Note expectations are generated much as in Larson's Multi-Level Model: the program makes predictions at a higher level (Layer 2) and then these predictions are used to make smaller-time-scale predictions on a lower level (Layer 1). Predictions are visible in the figure as green-colored notes. When a note arrives that is not expected, the program draws an orange exclamation point to indicate surprise. For example, in Layer 1, the program expected the note E, because it had noticed the pattern established in previous measures of notes moving up by scale step after the initial note of each measure. However, the note A occurred instead, so it drew an exclamation point above the A. Similarly, one quarter note earlier, in Layer 2, the program expected the note B to occur because of the perceived descending scale group **(E–D–C)**. (The green B is not shown in the figure because only the most recent wrong prediction, in this case the E, is displayed.) One might recall that Layer 2 is made of notes that have been perceived as structurally important, and one might thus wonder how a prediction can either come true or be denied in Layer 2, when the actual notes "heard" by the program are added to Layer 1, not to Layer 2. Well, in this case the note D was perceived in Layer 1, and subsequently it was promoted to Layer 2; this promoted note D was a surprise since the program expected B; therefore, the program indicated its surprise by the orange exclamation point above the D in Layer 2.

The division of the melody into separate measures (*i.e.*, the placement of bar lines) was *inferred* by the program, although this never worked very well. However, in the particular run from which this screenshot was taken, the dashed blue bar lines were *given* to the program along with the input; it was not attempting to generate bar lines during this run. Similarly, the key of the melody was supposed to be inferred, although development of this aspect never got very far; in later versions I provided the key as well as the bar lines.

Melodic motifs were not generated by this version of the program, but this mocked-up screenshot shows that they were intended to be displayed in a box at the right (labeled "Active Motives"). These are short melodic shapes that the program has heard as being salient. For example, the Layer 1 groups in measures 2 and 3 exhibit a 2-quarter-note pattern consisting of a simple note repetition. This pattern, if the program had actually noticed it, would have been designated as a motif: it is the third entry from the top in the "Active Motives" display.

MusicatH had a Slipnet modeled after the Slipnet in Copycat. This Slipnet included concepts such as "step", "leap", "ascending", "descending", "pitch repetition", "linear motion", "tonic", "subdominant", "dominant", and a collection of pitches such as "C", "D", "F♯", and "A♭". However, I didn't figure out a cogent way to coordinate the standard Slipnet mechanisms with the constant flow of time in the music domain, so I eventually (but reluctantly) removed the Slipnet. See Figure 9.3 for a screenshot of MusicatH's Slipnet.

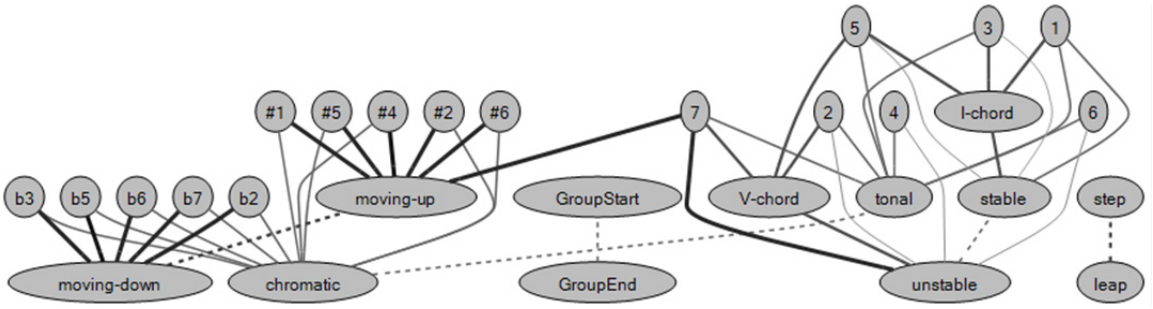

Figure 9.3: MusicatH's Slipnet.

An obvious difference between this version of the program and the most recent one is the use of standard music notation. Quite a large percentage of the code in MusicatH was devoted to the proper display of music notation. Correctly laying out elements of music notation is quite complex, even in the restricted case of Musicat's melodies. Writing my own display code allowed me to draw group ellipses that enclosed notes in an elegant way; see, for example, the way in which the large group in Layer 2 is tilted to slope downwards at the right end of the group, whereas the small group in Layer 1, measure 1, slopes upwards. Out of both speed and complexity considerations, I stopped drawing the notation so carefully in the more recent version of Musicat. See (Byrd, 1984) for more details on the fascinating topic of computer-generated music notation.

Example Runs with This Version

The snapshot above shows a fairly ***good*** run of the program (all things considered), and hence it masks a variety of problems. For instance, the program would often make groups in Layer 1 that seemed much less analogous to each other than these do. All three small green groups in the figure are similar to each other, as they start on beat 3, end on beat 4, and form the end of an ascending sequence, but this kind of similarity is in fact not very common.

A more obvious problem in this run is that many notes remain ungrouped. This is a serious problem — the first two notes, C and D, are not grouped, even though the promoted version of the C in Layer 2 is part of a group. In general, for most runs, many notes never become part of a group; this is because MusicatH did not feature a pressure pushing hard for each note to be seen as a member of some higher-level structure.

To give a better picture of how the program worked and to illustrate more of the problems it had on even the simplest of melodies, I present a collection of screenshots from MusicatH with minimal commentary. But why, one might ask, am I bothering to show and analyze results of a program that didn't work very well? The answer is that it is important to understand the extreme complexity of the domain. Musicat could not exist in its present form without our first having profoundly grappled with the many problems that came up in earlier implementations.

During the development of MusicatH, I spent most of the time experimenting with the Schumann melody from the previous example, with a "Triplet Scale" melody, and with a few others. In this section I will use just a few of these melodies as examples, as well as melodies such as "Twinkle, Twinkle, Little Star" and "Good People All", which I hadn't run the program on during development.

"Stückchen", by Schumann

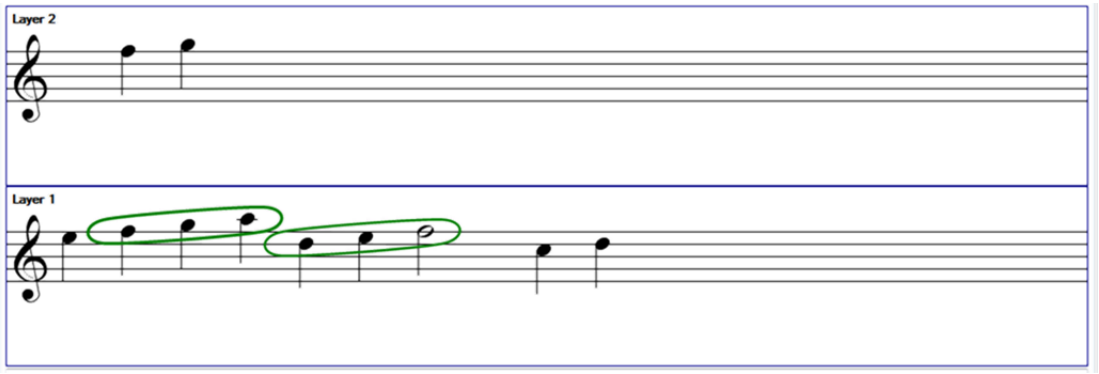

Figure 9.4: Stückchen, run 1.

In this run of MusicatH (Figure 9.4), I was using a version that had the ability to recognize half-notes and eighth-notes in addition to quarter notes, so I didn't have to break the half-note F into two notes as I did in the previously-described run. Notice that at this point in the run the first note, E, is ungrouped, and the two notes promoted to Layer 2, F and G, seem quite random and unrelated to any larger-scale linear structure of the music. At the end of this same run, however, things have changed:

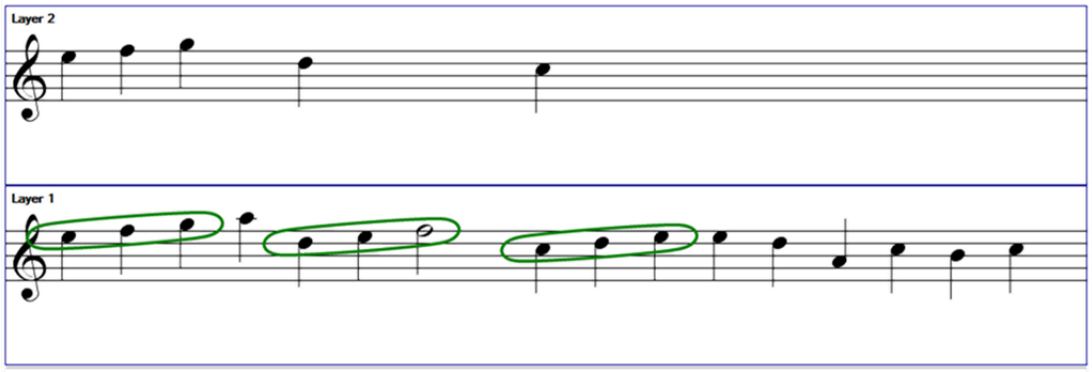

Figure 9.5: Stückchen, run 1.

The initial E in Layer 1 has become part of the initial group, but now the A on beat 4 is ungrouped. Such shifting back and forth between possible rival groups was very common in this version of the program, and it did not seem able to settle on a stable group structure with much reliability. Notice how many notes remain ungrouped here: all notes in Layer 2 and the final six notes in Layer 1 are ungrouped. One last point: the notes in Layer 2 make a little more sense: the E, D, and C are part of a descending scale. However, the notes F and G are extraneous, and they prevented the program from making the group **(E–D–C)**, which was present in Layer 2 in the previous run.

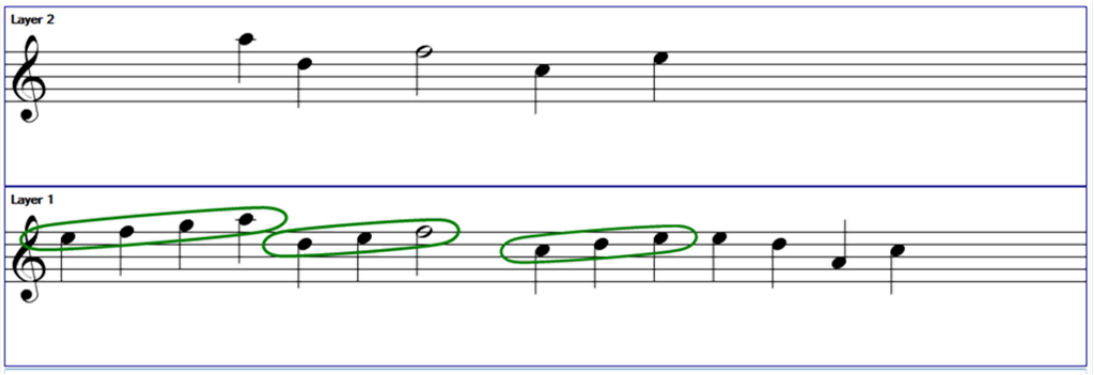

Figure 9.6: Stückchen, run 2.

In run 2, the group structure is quite understandable in Layer 1 (aside from the ungrouped final notes). However, the selection of notes to promote to Layer 2 seems mostly random (all of the promoted notes are the first or last elements of a group, but they do not function coherently to make linear structures in Layer 2).

Triplet Scale (Bach Organ Prelude in C major, BWV 547)

Recall the ascending scale-like figure discussed in Chapter 3:

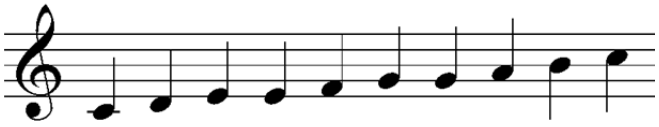

Figure 9.7: Triplet Scale.

When people initially hear a melody, they typically assume it is in a duple meter like 2/4 or 4/4, but given contrary evidence, they may quickly shift to an alternate metric interpretation. In this melody, people will switch to a 3/4 interpretation at some point (even if the notes are all played with exactly the same volume and articulation:

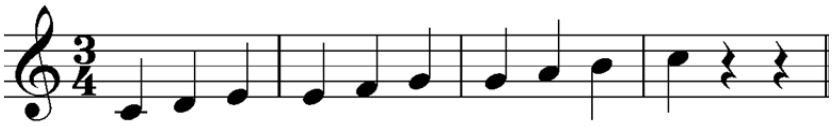

Figure 9.8: Triplet Scale, heard in 3/4 meter.

Let's take a look at how MusicatH "heard" this melody.

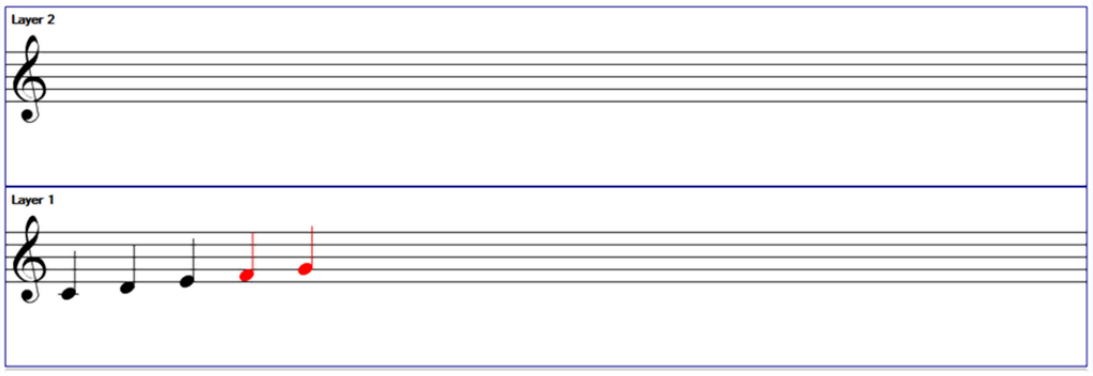

Figure 9.9: Triplet Scale, run 1.

After the first three notes (Figure 9.9), the program has generated an expectation (shown in red in this and the following diagrams) that makes sense: the scale is expected to continue up to G.

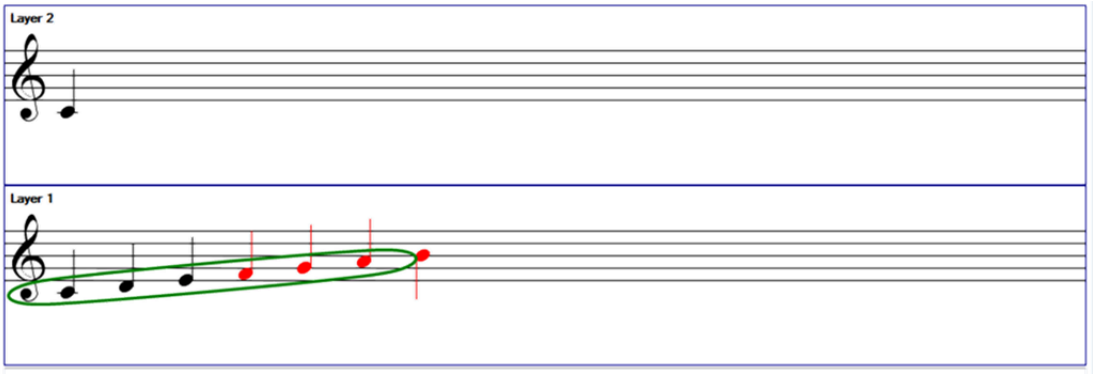

Figure 9.10: Triplet Scale, run 1.

A moment later (Figure 9.10), the prediction has been extended all the way through B, and the first six notes (not including the B, strangely) have been grouped together.

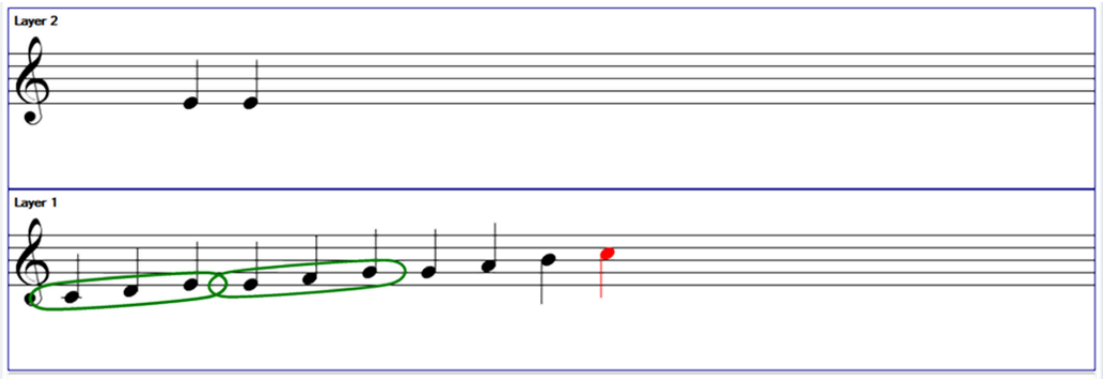

Figure 9.11: Triplet Scale, run 1 (end of run).

At the end of the run (Figure 9.11), the program's earlier predictions have been replaced with the actual notes. The first six notes have been grouped in way I expected: two groups of three notes each. The final three black notes, unfortunately, were not grouped. The program generated the obvious prediction for the scale to continue up to C (however, the final C wasn't included in the melody given to the program in this run.)

Layer 1, aside from the missing final group, showed a reasonable grouping structure. Layer 2, however, makes absolutely no sense. Ideally, the layer would contain an ascending

arpeggio, C, E, G, with a red expected note C at the end, just as in Layer 1. Obviously, the multi-layer mechanisms in MusicatH didn't work well.

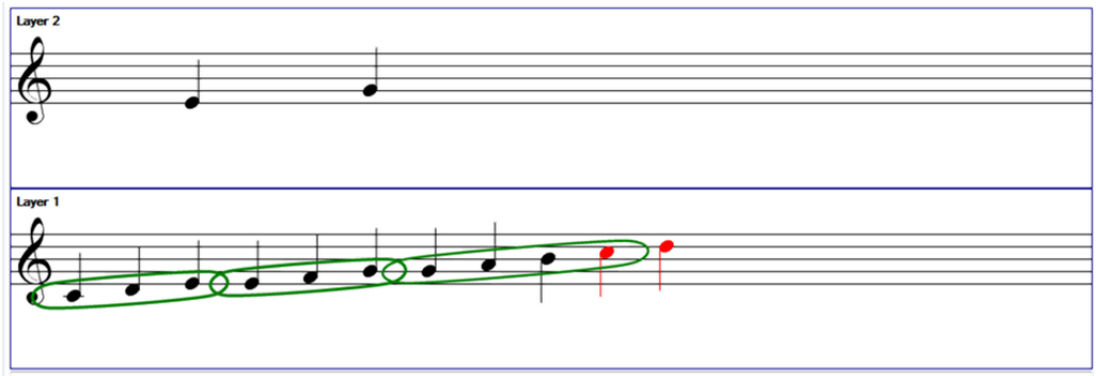

Figure 9.12: Triplet Scale, run 2.

In another run, the grouping structure in Layer 1 was quite reasonable. The final expected D, however, belies a lack of understanding of the melody (a repeated C would fit much better, if the program had understood the melody as based on an arpeggiation through a C-major triad in Layer 2). Layer 2 was better than in the previous run: it was on the right track, but the initial C was conspicuously missing, and thus no arpeggio was "heard" in the layer.

These two runs showed problems in Layer 2, but they did a passable job on Layer 1. However, many runs are quite worse. The next two show significant problems in both layers.

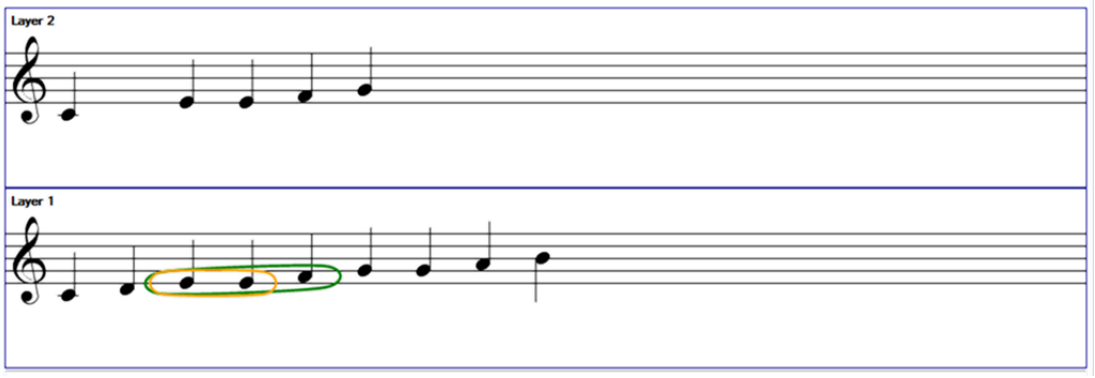

Figure 9.13: Triplet Scale, run 3.

In this third run, the grouping structure in Layer 1 seems almost entirely random. The program has made a pitch-repetition group (colored yellow) around the first two E's (such groups were missing in the previous runs — also notice that yellow and green groups are allowed to overlap), but the green group **(E–E–F)** makes little or no sense. Layer 2 also makes no sense; it is almost a copy of Layer 1 for the first six notes of the melody.

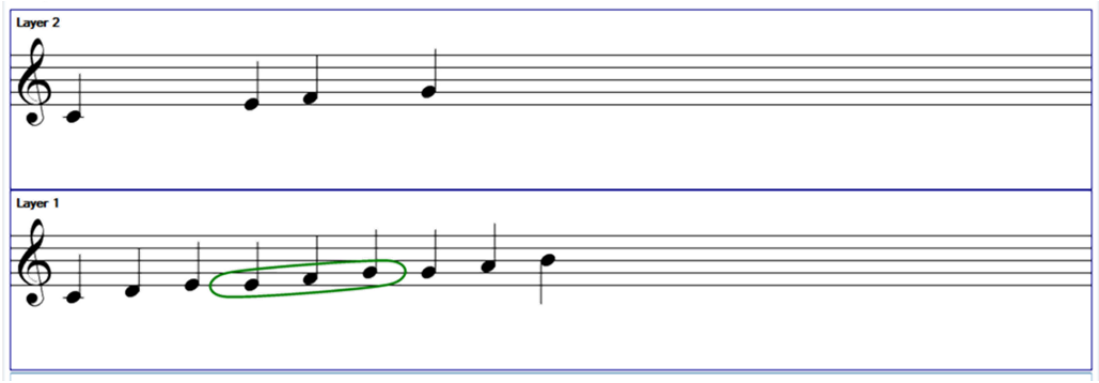

Figure 9.14: Triplet Scale, run 4.

In this final run, just one group has formed, in Layer 1. Layer 2 has the expected three notes C–E–G, but also an additional F that doesn't fit in. Altogether, very little structure has been found, and what little has been found makes very little sense.

Twinkle, Twinkle, Little Star

For this melody, the program notices many of the note repetitions — a very salient feature of this melody — but fails to understand anything much else of the structure.

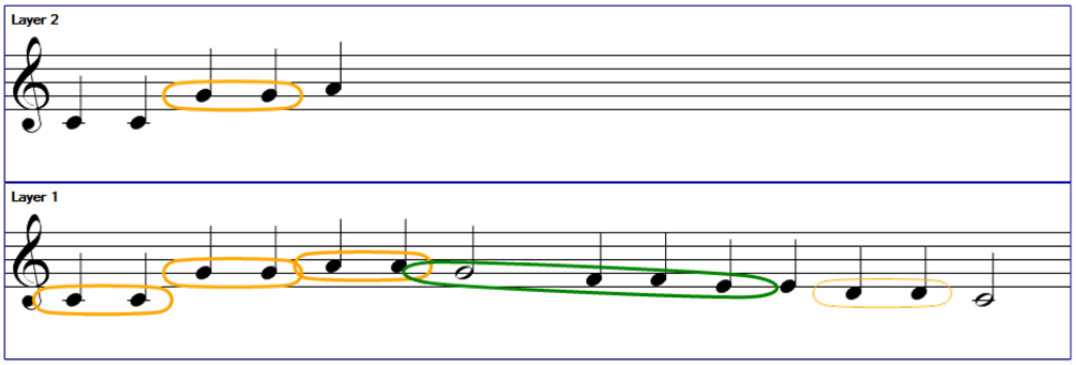

Figure 9.15: Twinkle, Twinkle, run 1.

The yellow pitch-repetition groups that exist are good, but there are some missing (such as around the two F's). Layer 1, again, doesn't make much sense. The green group captures some of the descending scale, but it's a very strange segment, going from the half-note G down through the first E. It is strange that it didn't continue to the final C, and also quite strange that the group *starts* on a long-duration note (the half note) that, to a human listener, *ends* the previous phrase. The program has noticed the linear descent, and it has essentially ignored the significance of the half-note duration of the note G. The pause in motion created by the half note strongly influences group-creation for a human listener and suggests that a group should end just after this pause. The problem of ignoring the group boundary implied by the half-note was a central issue for the next version of the program (Musicat without Hierarchy), when running on the melody "On the Street Where You Live".

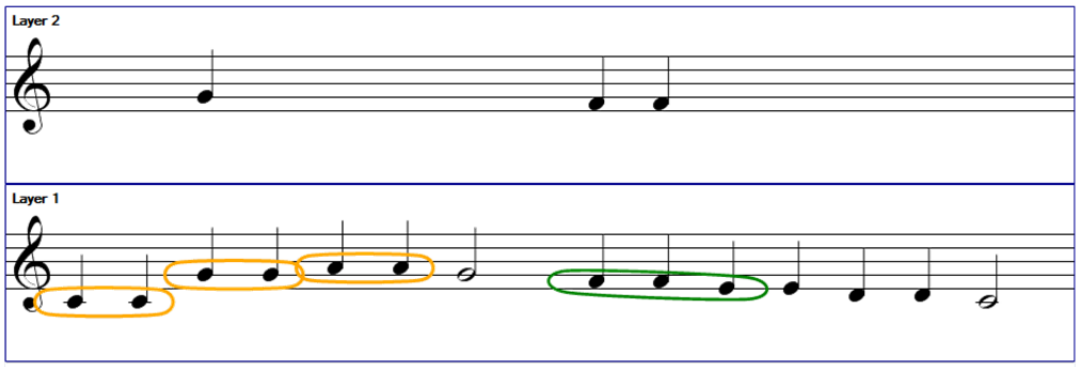

Figure 9.16: Twinkle, Twinkle, run 2.

In a second run (Figure 9.16), similar structures formed, but the green group was shorter…

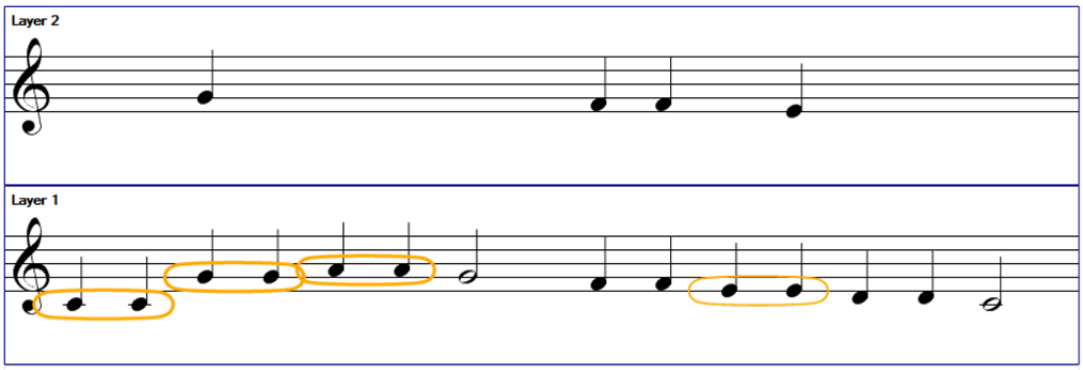

Figure 9.17: Twinkle, Twinkle, run 2, a moment later.

… and just a moment later (Figure 9.17), it had disappeared entirely (although another yellow group has formed). This gives a taste of another problem that plagued the next versions of Musicat (and that continues to cause trouble in the current version): the program is flexible in trying out different structures, but far too often it seems to "flail around", trying out various small structures without seeing the larger structure. In this case, the program never notices the long descent from F (or even G) to the final C: it tried out the short **(F–F–E)** group and then gave up, thus failing to perceive any good large structure.

Good People All

A positive thing about the early versions of Musicat was that they could make groups at any point. In "Good People All", this feature allows one group to form that is impossible in the current version of Musicat:

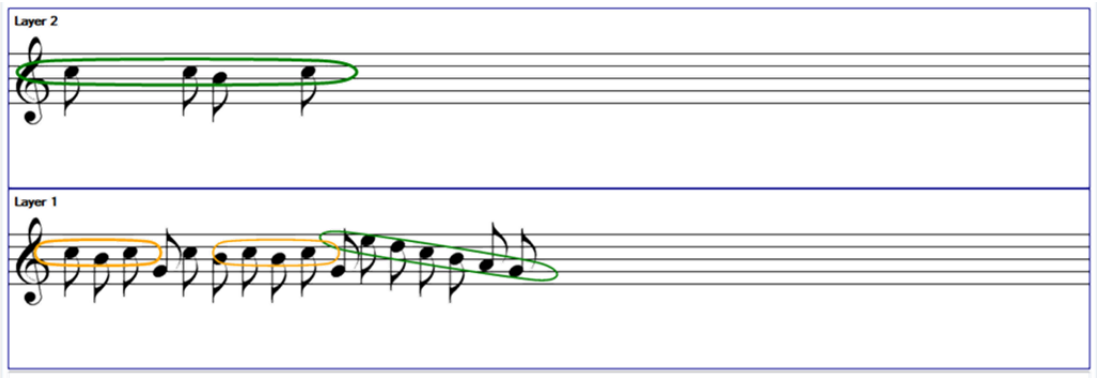

Figure 9.18: Good People All (first two measures).

The green group in Layer 1 shows that the final six notes of this excerpt have been heard as a descending scale, and this segment is separate from any other groups. Unfortunately, the program did not make any groups for the first ten notes, aside from the two yellow pitch-similarity groups, which exclude not only the non-C notes, but also the third C, which is ungrouped. The green group in Layer 2, however, does appear in the right place — I might hear the first ten notes as forming a single group, and the program has picked the three most important "C" notes (in my own hearing) to promote to Layer 2. It is unfortunate that a stray B was promoted as well, and also that the extremely salient E, which starts the descending scale in Layer 1, was not promoted to the top layer.

On the Street Where You Live

I will show more examples of this melody in the section on the next version of Musicat, but it is especially instructive to see a problem that crops up in this early version of the program on the first four measures (notice that none of these examples have been more than a few measures long, because the program did extremely poorly with longer ones):

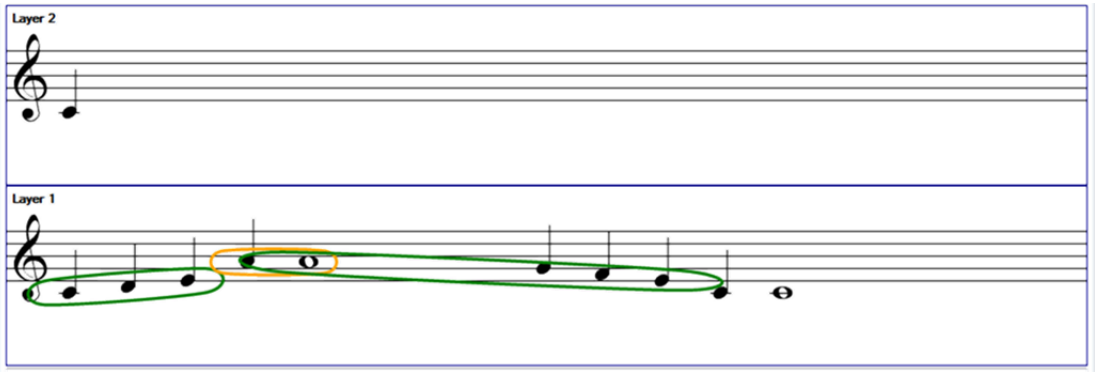

Figure 9.19: On the Street Where You Live, run 1.

The first run on "Twinkle, Twinkle" (Figure 9.15) exhibited a very similar problem, but it is more pronounced here: the second green group starts, quite unnaturally, on a long note, the whole note A (the program's display routine draws the group so that it extends too far to the left and touches the quarter note A, but for the program, the whole note A is indeed the first note of the group). This problem of starting a group on a long note was quite persistent and was very hard to eradicate for quite some time during the development of Musicat. Why did the program create this bizarre, long group that so obviously spans the huge temporal gap between the whole note A and the following G?

There are two obvious competing factors in the grouping here. First, the whole note creates a pause in the previously established rhythm of note attacks (there had been an attack once every quarter beat, and suddenly there are four beats between attacks). This pause creates a pressure for a group boundary to form after the long note — this group boundary is

obvious to me, and I believe that any human listener would form the boundary at this location. A second relevant pressure, however, exists in the program: it favors making groups exhibiting stepwise motion, and longer groups of this sort are assigned higher strengths. The stepwise descent from A to E is long indeed, and hence results in high strength value. To human ears, however, the temporal gap in attack onsets caused by the whole note is much more salient, and the program's grouping seems ridiculous.

It would be possible to adjust the relative strengths of these two pressures to fix this problem, but there is another factor that helped even more to reduce this problem in subsequent versions of Musicat: analogy. The first five notes should be heard as *analogous* to the next five notes (*i.e.*, the first two measures are analogous to the second two, if we have bar lines and a 4/4 time signature). This parallelism makes the grouping boundary after the whole note even more obvious, because the three quarter notes G–F–E are so easily mapped onto the initial notes C–D–E.

The next example (Figure 9.20) is of a moderately good run that illustrates a subtler problem in the same melody:

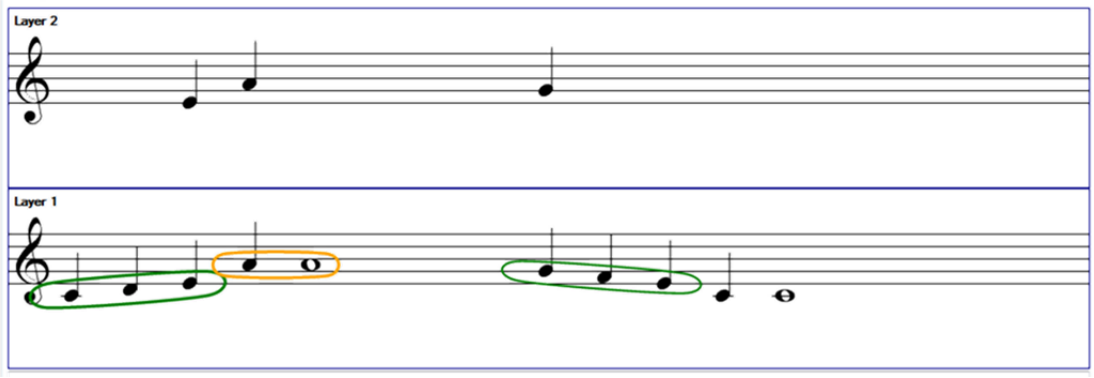

Figure 9.20: On the Street Where You Live, run 2.

Both of the three-note scale passages have formed groups (shown in green). The first note repetition has also been identified (shown in yellow). Unfortunately, the matching final note-repetition (the two C's) did not yield a group. Layer 2 has picked out some potentially-important notes, but absolutely no large-scale structure has been discovered.

Perhaps most importantly, in terms of the development of Musicat, notice that this version of the program doesn't make group links in the display. Behind the scenes, while the program is running, codelets do use the existence of one group (such as the first green group) to inspire a search for similar groups (such as the second green group). However, once a similar group has been formed, no analogy is perceived between them. The loss of this linking information severely crippled this version of the program. If an analogy had been established between the green groups, it could have guided the search for a *second* yellow group, and a run that started as shown above could have resulted in a complete grouping structure, at least in Layer 1.

I gave examples of the centrality of analogy-making at all levels in Chapter 4, but ironically, during Musicat's development, analogy wasn't a large part of the program until relatively late in its development. Explicit group links, which could indicate that two groups were similar in some way, were part of the program already since 2008 (Musicat Without Hierarchy), but a more structured "analogy" object, which could describe the mapping between two sides of a correspondence, was not added to the program until early 2011. In retrospect, this sort of analogy object — one that could collect together a set of relationships amongst its components — seems to have been one of the most crucial missing pieces in these early versions of Musicat.

Musicat Without Hierarchy (MusicatW)

It is obvious from MusicatH's results in the previous section that Layer 2 wasn't useful, mainly because there was not enough structure perceived in the notes that were promoted to it. While I believe that this multi-layer idea has much merit, it would have required significant changes to make it work well. Rather than spending more time on that aspect of the program, I decided to remove the multiple layers of hierarchy. The resulting version, Musicat Without Hierarchy (henceforth "MusicatW") may have lacked a Schenkerian-style hierarchy, but in its place I added the ability to form meta-groups, which provide a different way to look at hierarchical structure. Some aspects of musical structure, such as linear motion on a large time-scale, are harder to capture without a multi-layer analysis, so I think it would be useful to re-examine this idea in the future, once the program works much better on smaller structures.

MusicatW evolved in design quite a lot over the two-and-a-half years of its development, starting with the removal of hierarchy layers and ending with the addition of a new "analogy" object type. I will describe the essential elements of the final form it took in 2011 before the most recent major redesign.

Main Features of MusicatW

MusicatW was the most complicated version of Musicat, because over the course of its development many different features were added, both in the model's architecture and in the user interface. Figure 9.21 is a screenshot of MusicatW in action for nearly the same melody as was used to introduce the previous version (the only difference is that here, the F in measure 2 is a half note instead of two quarter notes, as in the later runs of MusicatH in the previous section).

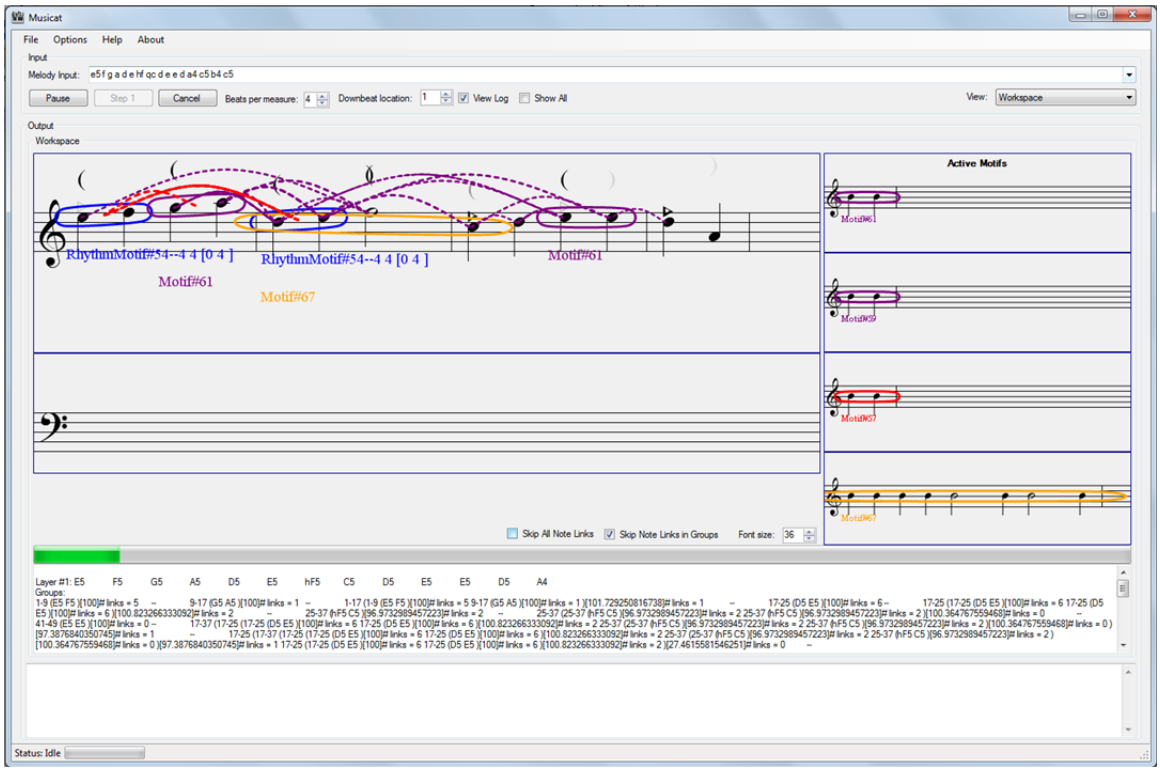

Figure 9.21: MusicatW running on "Stückchen".

Notice a few brand-new elements here: there is only one layer — the melody itself, in the top staff, but there is also a staff displaying an optional bass line (not used in this example). There is a short green bar underneath the bass staff. The length of the green bar indicates the current global temperature of the Workspace.

In this screenshot, the "Active Motifs" are actually generated by the program, as are all the links between structures in the Workspace. Dashed *red* arcs connecting *groups* are group links, whereas dashed *purple* arcs connecting *notes* are note links. In MusicatW, note links take the place of the yellow pitch-repetition groups generated by MusicatH, and are also more general (notes heard as similar do not need to share the same pitch: a D in one measure might have a note link connecting it to a G in another measure. Note links tend to

clutter up the image, so the following figure shows the same run from a moment earlier in the run, without the note links. I have also zoomed in to display only the Workspace.

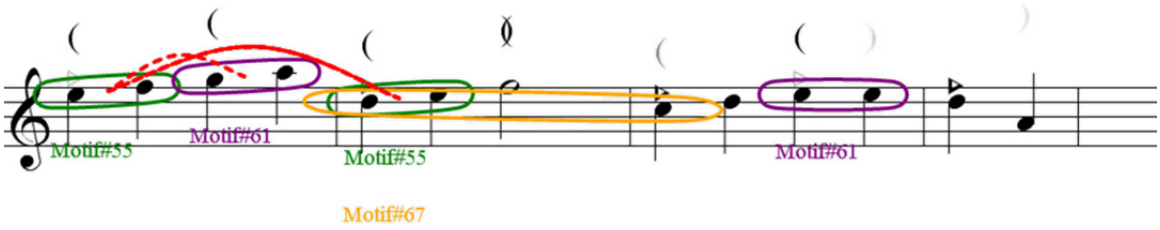

Figure 9.22: MusicatW running on "Stückchen", note links hidden.

Several elements unique to this version of Musicat are visible in Figure 9.22. These include motifs and note perspects.

The program looks for short repeating melodic patterns, and when it finds one, it calls it a "motif" and assigns it a unique identifier[16] such as "Motif#55". For example, in the figure, there are two groups labeled "Motif#55" and colored green. These groups share a rhythmic pattern and melodic contour: they both start on beat 1 of a measure, contain two quarter notes, and consist of a single step upwards. All of these details are part of "Motif#55". A similar distinct motif in the figure is "Motif#61". The two purple groups in the figure have been labeled "Motif#61", but they are slightly different from each other in contour. Each purple group starts on beat 3 and is comprised of two quarter notes, but the first one has an ascending-step contour (G–A) while the second one has a flat contour (the note E is repeated). In this example, I looked at a separate display in the program to discover

[16] Motif ID numbers, somewhat surprisingly, are always greater than 50 or so. This does not mean that there were 50 other motifs generated by the program earlier in the run (or previous runs). Rather, the motif number is set to be the same as the ID number of the node in the Slipnet that represents the motif (motif nodes in the Slipnet are discussed below). The Slipnet contains approximately 50 nodes before motifs are added; therefore, motif numbers start in the 50s.

that the standard form of Motif#61 has the flat contour, like the second purple group. However, the program has some flexibility in deciding whether a group is an instance of a motif, and in this case it assigned the **(G–A)** group the "Motif#61" label even though its contour was not the contour expected by the motif — the *rhythm* of the group was exactly what *was* expected, and this was good enough. In assigning both groups this label, the program has made a sort of analogy between the groups. Unfortunately, however, an explicit group link wasn't created in this run between these two groups (a link was created to join the two green groups, however).

Behind the scenes, a motif such as "Motif#61" is represented by a "motif node" in the Slipnet. These special types of Slipnet nodes are created and added to the Slipnet by motif-creation codelets. A motif such as Motif#61 also has links to two other motif nodes: one of these describes the rhythmic motif involved, and the other describes the contour motif. Motifs based on exact pitches (instead of contours) or on sequences of "note perspects" (see below) were also possible in the architecture[17], but there were no codelets that actually created motifs of these sorts. Because they are in the Slipnet, motif nodes can become activated, just like any other Slipnet node, and activation can spread through the graph of motif nodes. An activated motif node triggers codelets to search for more instances of that motif. This mechanism worked well, in my opinion, and it should be restored in future versions of Musicat. Although motif nodes are stored in the Slipnet, I find it simpler to refer to the collection of all motif nodes as the "motivic memory". The motivic memory can also be saved at the end of a run and used in future runs. This allows for a limited type of learning in Musicat: the motivic memory may be extended in each run to include more motif nodes,

[17] This system of representing a motif as a combination of patterns along several different musical dimensions, and of seeing motifs as nodes in a conceptual network, has much in common with Olivier Lartillot's work. See, for example, (Lartillot, 2002, 2004, 2005).

and in this way the program can in principle recognize motifs in one melody that it originally heard in another melody.

In the program's interface, groups are labeled with at most one motif, but internally, groups can be associated with multiple motifs, each one with a different strength. The display shows simply the strongest motif for each group. The collection of multiple motifs for one group is implemented using the general *perspect* mechanism that is unique to this version of the program. I borrow Ockelford's term "perspect" to describe any parameter of a note or group that has been noticed by the model (recall from Chapter 2 that the word "perspect" comes from the phrase "perceived aspect"). Each note or group has a predefined set of weighted perspects[18]. These include "group start", "group end", "harmonically tense", "tonic", "dominant", "leading tone", "salient", and so on. For instance, if a bass line has been given along with the melody, and if a codelet notices that a note has high harmonic tension with respect to the current bass note, then the codelet may add weight to the "harmonically tense" perspect for the note. Motif perspects are added dynamically: if a codelet notices that a group is similar to a motif stored in the motivic memory, then that codelet may add a motif perspect to the group, setting the weight of the perspect according to the degree of similarity between the group and the standard form of the motif. Perspect weights can be modified by other codelets. In some experiments I made the weights decay over time, although this generally led to too many perspects disappearing completely while they were still needed.

Three perspect types (in addition to the motif labels) are visible in the figure above. There are small right-pointing triangles that indicate perceived stressed on notes (*e.g.*, over

[18] Although I use Adam Ockelford's terminology (Ockelford, 2004), Steve Larson deserves credit for the idea of having Musicat associate a list of qualities with each note. He suggested to me (personal communication, 2008) that each note in a melody "has its own personality", and that Musicat should make a list of the different qualities of each note in context.

the final D in the melody). The strength of the perspect is represented by the color of the marking, ranging from black (for a strong perspect) to light gray (for a weak perspect). The other types of perspects visible are the "group start" and "group end" perspects, represented by left and right parentheses, respectively. For example, the first note in the melody has a strong "group start" perspect, indicating that the first note sounds like a likely place for the start of group. The final E in measure 3 has a weak "group end" perspect. Perspects are not mutually exclusive: the half note F in measure 2 has both group perspect symbols, meaning that it has been heard as a likely place both for a group to start and for a group to end.

Althought the perspect and motif mechanisms have much potential, they did not survive in the latest version of Musicat because they were focused on perception of very small structures (at the scale of one note or a few notes), while the latest Musicat was focused on perception of larger structures. I expect to resurrect these ideas in future incarnations of the program.

Another element implemented in MusicatW but not in other Musicats is global temperature. For example, the latest version of Musicat has only *local* temperature. I believe that global temperature makes more sense when it is applied to a small structure (such as one that can be stored entirely in working memory) than to a larger structure, such as a 16-measure melody, that cannot be perceived in its entirety at one moment. Figure 9.23, below, shows a plot generated by the program of global temperature as it changed over the course of a run.

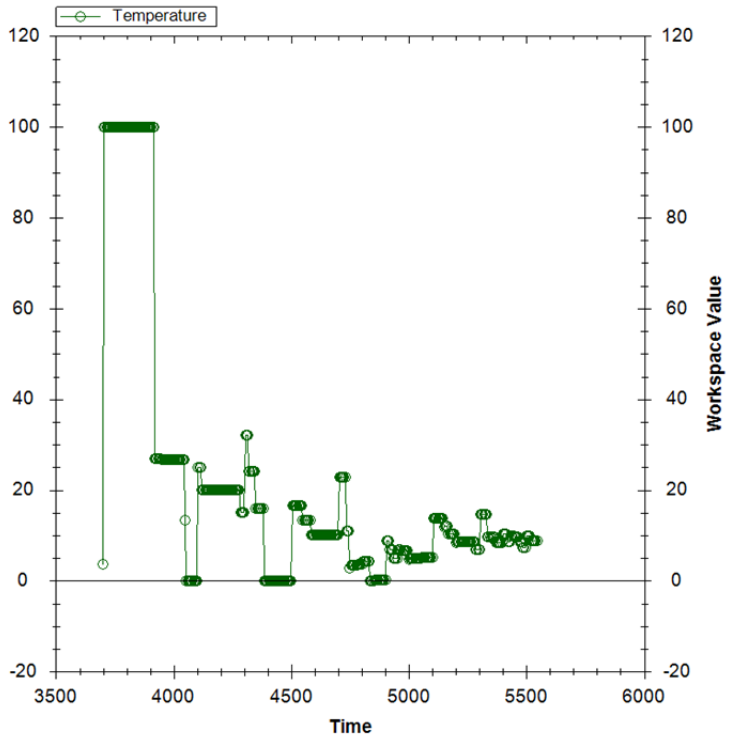

Figure 9.23: Global temperature.

In MusicatW, it is possible to add to this graph a set of lines representing the urgencies of codelets of various types. These graphs were quite useful during development, as they helped me understand how codelets were behaving during runs.

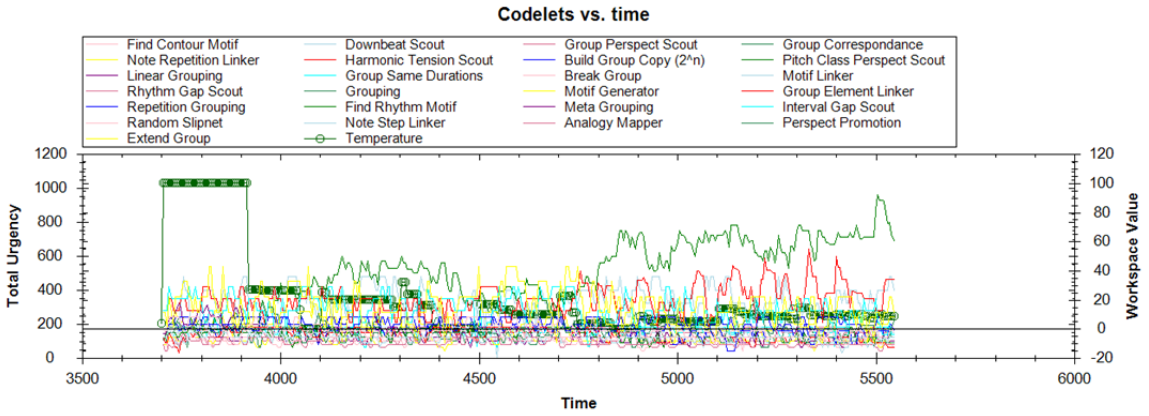

Figure 9.24: Codelet urgencies during a run.

Figure 9.24 shows the total urgency for every codelet type that was active during one run. It contains far too much informations, and so it is hard to glean much insight from it. More typically, I would include just two or three codelet types in the graph at one time, which made it much more comprehensible and hence useful to me.

Figure 9.25: Real-time codelet adjustment window.

Figure 9.25 shows a part of the user interface that I created to help during development. It is similar to the graph in Figure 9.24 in that it lists all the codelet types and displays the total urgency of each. Here, total urgency is represented by color: high total-urgency codelet types are shown in red, while low total-urgency types are shown in green. The sliders indicate the default urgency value for new codelets of each type, and can be adjusted while the program is running. Additionally, clicking on the name of a codelet type in this window will post more codelets of that type (using default parameters) to the

Coderack. I used this to experiment with the balance between different codelet urgencies and to better understand the behavior of the program as it ran.

So far I have focused on new features of MusicatW that were visible in the user interface: note links, group links, motifs, perspects, global temperature, and displays of codelet urgency. Many other features affected the program behind the scenes. The following list describes a few of the most important such features.

Key Signature and Meter

Whereas MusicatH had the goal of automatically inferring the key and meter of a melody, in MusicatW these were given to the program, so that it would be freed up to focus on other aspects of listening. Key and meter inference is an interesting aspect of listening, and one that I think this architecture would be good at modeling. However, in a real listening scenario, there are many cues relevant to key and meter that are available to a human listener but that are not part of Musicat's input, such as stress and phrasing. Thus, the listening challenge given to Musicat without the key and meter information (*e.g.*, listening to unaccented notes in perfect rhythm with no expressive timing) was perhaps harder than it should have been.

Pitch-contour Wildcards

Nodes in the motivic memory describing melody contours are very flexible, compared with how contours are represented in other versions of Musicat. A contour consists of a sequence of transitions in pitch between adjacent notes, such as "leap up — repeat pitch — step down", just as in the latest version of Musicat. However, in MusicatW a transition can also be described with a wildcard symbol, '*', indicating that any direction of

movement is acceptable. For instance, in the contour above, the "repeat pitch" element could be replaced with '*', leading to the contour:

leap up — * — step down

This contour is more general than the previous one, and therefore more melody fragments could match it. A motif incorporating this contour would be a more general motif. Muiscat could theoretically (that is, with additional codelets) come up with generalized contours of this sort when necessary for a particular melody. However, I explored this idea only a little because I decided to concentrate more on analogy-making in the Workspace than the behind-the-scenes analogy-making that would be involved in getting Musicat to make these more-general motifs. More importantly, the notion of trying to specify a musical motif in such a formal and mathematical manner — such as that suggested by lists of transition types and possible wildcard symbols — is not in the spirit of the fluid perception that is a goal of the FARG architecture. (It is ironic that the addition of the seemingly *flexible* wildcard character was precisely the thing that led to my realization that this sort of contour description was too *rigid*.) Olivier Lartillot's computer models of melodic motif extraction have similar goals as MusicatW, and he reached the similar conclusion that "musical patterns actually discovered by the listeners cannot be reduced to simple mathematical objects"(Lartillot, 2005).

Context-based Scoring of Structures

Some of the challenges associated with computing the strength values for various structures in the Workspace were already discussed in Chapter 8. In MusicatW I experimented with several different scoring strategies mentioned in that chapter, but the

most effective technique I used during MusicatW's development was the idea of running statistics that I implemented in Paris in 2010: group scores were computed based on a weighted sum of a set of various criteria, and these scores were then compared with the scores of other groups scored by other codelets. The mean and variance of scores in a recent time window were used to determine how a newly-computed score stacked up against other scores. This allowed greater flexibility in designing the scoring function and choosing weights for its various features. However, I eventually stopped using this system because it was very complicated to understand the program's behavior when the strength of one structure changed rapidly based on the computed strengths of other structures.[19]

Tabu Search

MusicatW has many numerical parameters that I had to choose using my own intuition and experimentation. I added elements to the user interface to make it easy to modify parameters, and even to selectively disable functionality (*e.g.*, individual codelet types can be disabled, as can individual terms of structure scoring functions). This was useful in experimenting with parameters by hand. After all these parameters were exposed via the user interface, I took the additional step of making a command-line version of MusicatW that could be run by a separate program (with the user interface disabled). This external program could specify the parameter values to use in a run. The command-line version of Musicat would quickly run and then report the final state of the Workspace. Thus, runs of MusicatW could be automated, which opened up the possibility of automatically tuning the program's

[19] Structure scoring based on a weighted sum of features is a very important but not well-enough understood aspect of FARG models, in my opinion. The problem of *learning* feature weights came up in my work on automatic generation of piano fingerings (Kasimi, Nichols, & Raphael, 2007) and I worked on a solution applicable to that domain (Raphael & Nichols, 2009), but for FARG models, the question of how to best approach these issues remains open.

parameters: given a set of sample melodies and desired grouping and analogy structures, the parameters could be modified automatically to try to optimize performance.

Because MusicatW is so complicated, not to mention stochastic, I considered optimization techniques that treat the function to be optimized as a "black box". In this case, the function in question was the error obtained by performing a complete run of MusicatW on a melody and comparing the groups and analogies of the resulting Workspace with the desired groups and analogies that I specified, and then repeating the process for several runs and averaging the result, to account for MusicatW's variability between runs. I chose the technique of "tabu search", which does randomized hill-climbing search in high-dimensional spaces and requires no knowledge of the form of the function that is being optimized (Glover & Laguna, 1997). The "tabu" part of the name refers to a heuristic for avoiding getting stuck in local maxima: in brief, if a parameter is modified during an iteration of the algorithm, it cannot be modified again until several other parameters have been changed during later iterations.

Tabu search, once it was implemented, offered small, incremental improvements in MusicatW's results on a small set of text melodies. However, the improvements were not very significant, and I had the strong impression that the problem was that the architecture of the program was not possible of doing significantly better, for *any* collection of parameter values — what was needed was far more substantial a change. In particular, MusicatW could not make "analogy map" structures (see below) at the time that I implemented Tabu search — those were added later.

Bar-line Thickness

I first implemented the idea of bar lines with variable thickness towards the end of the development of MusicatW (in most of the example runs below, thick bar lines are not visible because most of the runs were done with a slightly earlier version). The program preferred to create groups that would start or end just after or just before a thick bar line (with thickness dependent on group length), and to avoid crossing thick bar lines. Bar lines in this version are generated in a perfectly regular manner: the thickness of a bar line is proportional to the largest power of two that evenly divides the measure number of the preceding measure. In other words, measures 2 and 6 are followed by a bar line of a certain thickness, measures 4 and 12 are followed by thicker bar lines, measure 8 is followed by a bar line that is thicker still, and so forth. For example, recall the figure from the previous chapter, repeated here (Figure 9.26).

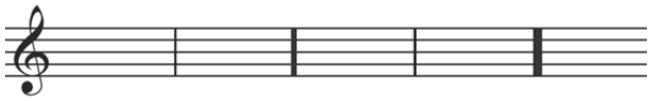

Figure 9.26: Bar lines of "thicknesses" following a regular pattern.

In the latest version of Musicat, bar lines *tend* to conform to this regular pattern, but there is more flexibility. In MusicatW they are *forced* to follow this pattern.

Analogy Maps

The idea of explicitly representing structured analogies (at various hierarchical levels) between groups (at various hierarchical levels) did not appear in Musicat until close to the end of development of MusicatW. In earlier versions of the program, analogy was represented in a less structured and more implicit manner, through mechanisms such as note links, group

links, and motif labels. The "analogy map" object type was added to the program to make explicit the set of relationships between two objects.

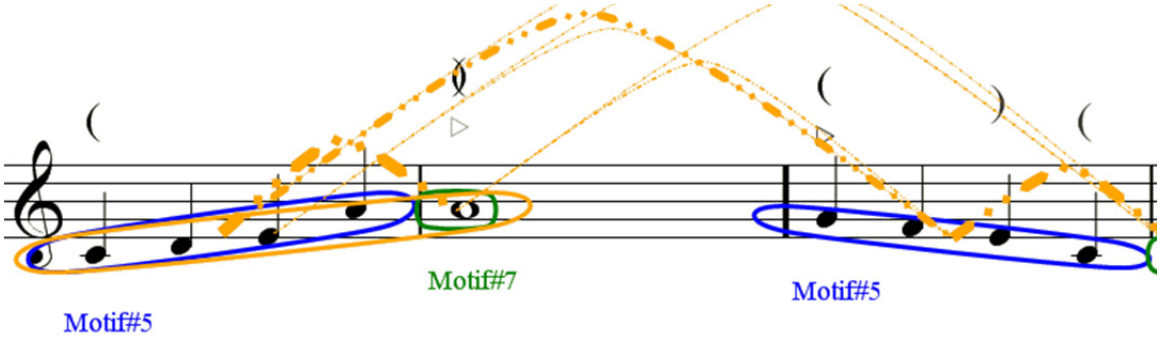

Figure 9.27: Analogy maps in MusicatW.

In MusicatW, analogy maps (or simply analogies) are displayed as thick yellow arcs with a pattern of alternating dots and dashes. Figure 9.27 gives an example in which several analogies are visible, including the admittedly strange analogy between the groups in measures 1 and 2, and a more understandable analogy between the groups in measures 1 and 3 (both groups are labeled "Motif#5").

As was the case with bar lines, most of the examples below do not include analogies because analogies were added late in development. Analogies were much better developed in the latest version of Musicat.

Example Runs With This Version

This section presents several screenshots of MusicatW running on most of the melodies that were used as examples above for MusicatH. Two additional melodies are included at the end of this section to demonstrate some significant problems with MusicatW that motivated the major changes in the next version of the program.

The Problem of "Flailing"

In general, the examples below are primarily intended to show how much trouble MusicatW had on these short and simple melodies, even though MusicatW was a quite sophisticated program that was the result of several years of development and improvement. The program's difficulty with these melodies illustrates both the complexity of the problem of modeling listening and the insufficiency of MusicatW's architecture. Watching the program running was often frustrating, because it would sometimes start to make "good" groups and discover appropriate descriptions of motifs, but then it would proceed to make many "bad" groups as well. When Douglas Hofstadter was watching early versions of MusicatW running, he would often describe the program's behavior during a run as "flailing". During the time we were in Paris in 2010, he delighted in finding ever more colorful French words and idioms to describe the program's ever more frustrating flailing problem, including:

- *ramer*: to row (but getting nowhere);
- *piétiner*: to be at a standstill, to stamp one's foot;
- *galérer*: to work hard, to slave away;
- *tourner en rond*: to go around in circles;
- *pédaler dans la choucroute*: to pedal in sauerkraut

My goal was always to stop the program's "flailing", but this was not successful at all until the latest version (and even the latest version flails to some extent, but the situation has been much improved.)

"Stückchen", by Schumann

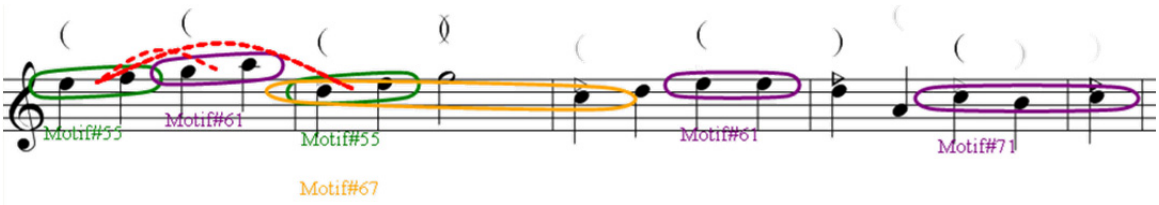

Figure 9.28: "Stückchen".

Figure 9.22, above, showed a snapshot of an earlier moment of the run depicted in Figure 9.28, but I will offer a few more comments here. The program has made several short groups of only two notes each. In general, I hear longer groups in this melody. The shorter groups make it easier for the program to find repeating patterns, such as the two instances each of Motif#55 and Motif#61, but they also reduce the amount of larger-scale analogy-making that occurs. For example, I hear measure 2 as analogous to measure 1, because both feature a stepwise ascending pattern. Measure 2 sounds like a version of measure 1 that has stopped short for a brief moment of repose on the dominant, G. MusicatW, however, "hears" only the connection between the first halves of these measures. When it does make longer groups (and meta-groups), they are sometimes hard to justify, as in the yellow meta-group that starts in measure 2, beat 1, and continues to include the downbeat of measure 3, instead of stopping after the half-note G, as I would expect. Relationships between groups are also lacking: the two red relationships in measures 1 and 2 seem good, but the rest of the melody has no relationships — not even between the two instances of Motif#61!

The group-start and group-end perspects above the staff are interesting in this run. Most of the notes that were on a downbeat or on beat 3 were heard as likely places for a group to start, as is indicated by the '(' symbols. This was a result of the program hearing beats 1 and 3 as having accents (this is built into its knowledge of common-time meter,

which has 4 beats per measure). There is a very faint, difficult-to-see group-start symbol above the A in measure 4 (beat 2); although the note A remains ungrouped, it would make some sense, to me, for a group to be heard starting here. This would be coherent with the previous note, D, being heard as the end of group. The D is also ungrouped, but it has a group-end perspect associated with it, indicated by the ‘)’ symbol, so it seems that the program was on a reasonable track in this case, even though the actual grouping structure failed to include either of these notes.

Triplet Scale

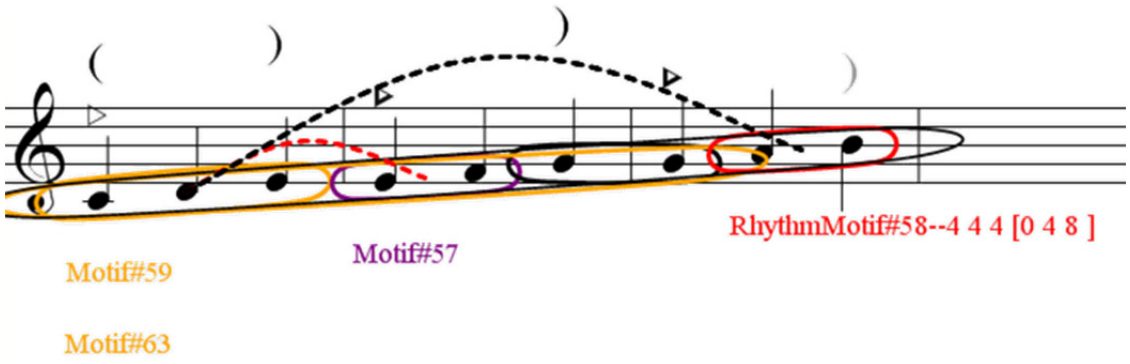

Figure 9.29: Triplet Scale.

This run on the simple “Triplet Scale” is simply terrible. It starts out well with a group of three notes in measure 1, but after that it makes three groups of two notes each: **(E–F)**, **(G–G)**, and **(A–B)**. The group **(G–G)** is central to the problem here. According to a human listening performance, this melody is made up of three groups of three notes each, but the end of each group (except for the last one) is linked via pitch repetition to the start of the next group. For example, the group **(C–D–E)** ends on E and the group **(E–F–G)** starts on E, but unfortunately, this latter group did not form in this run. The group **(G–G)** was in competition with the three-note groups **(E–F–G)** and **(G–A–B)**, preventing either from forming and persisting through the end of this run.

During the course of any given run, I would typically see the program creating and destroying various combinations of the expected three-note groups and rival two-note groups such as **(E–E)** and **(G–G)**. This behavior exemplified the "flailing" mentioned above.

For this melody, the program would benefit greatly from understanding how the end of one group can link (via pitch repetition) to the start of the next group. These kinds of relationships are important in many other melodies, but none of the versions of Musicat (including the most recent one) pay much attention to such relationships.

Twinkle, Twinkle

For the melody "Twinkle, Twinkle, Little Star", MusicatH recognized many of the repeated notes but did not form any useful larger-scale structures. MusicatW's listening performances were slightly better, as I will show with a few runs below.

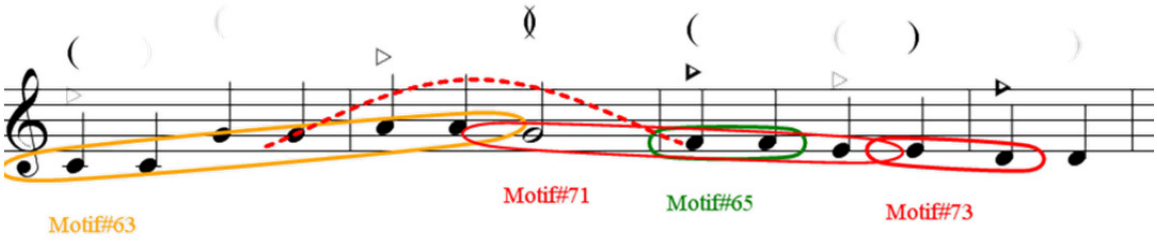

Figure 9.30: "Twinkle, Twinkle" (run 1).

This first run is not a particularly good run. One improvement over MusicatH, however, is that most of the notes have been incorporated in *some* group or other. Unfortunately, the groups do not make much sense, so I won't even attempt to explain all the strange groups formed here. I will just point out that the G in measure two is the start of a large red meta-group, when it should instead be the end of the large yellow group that comes before.

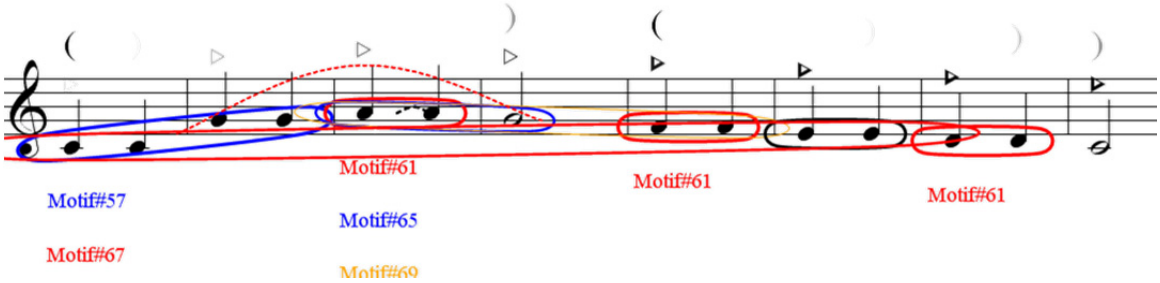

Figure 9.31: "Twinkle, Twinkle" (run 2).

In this run and the next, I told the program to interpret the music in 2/4 instead of 4/4. This run resulted in the program finding a 2-note pattern, Motif#61, which shows up in three places. Strangely, however, that motif was not perceived in measures 1, 2, or 6. Again in this run, most notes were group members, but the longer groups and meta-groups make little sense.

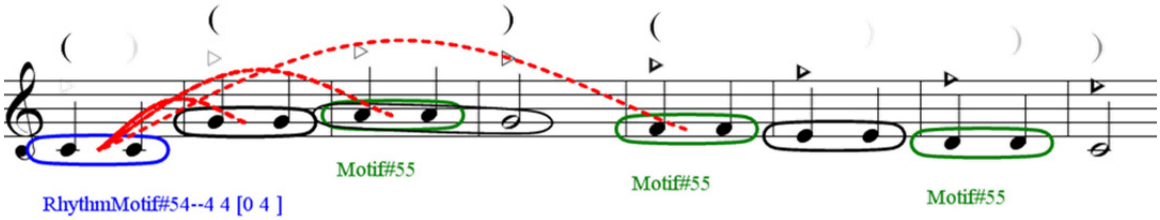

Figure 9.32: "Twinkle, Twinkle" (run 3).

This final run is very similar to the previous one. Notice the blue group at the start, labeled "RhythmMotif#54". This indicates that the group was heard as an instance of a particular rhythmic motif, but that the pitch aspect of the group was not heard as motivic. In this example, it is very strange that the blue group is not labeled as "Motif#55", because Motif#55 has a rhythm consisting of two quarter notes and a trivial contour consisting of a "sideways" motion (pitch repetition). Measure 1 satisfies both of these conditions, but due to the stochastic nature of the program, only its rhythmic aspect has been perceived (and indeed, three group links were made, all originating from measure 1 and linking to measures 2, 3, and 5, respectively), while its contour aspect has been ignored.

On the Street Where You Live

Recall from Chapter 7 that for the melody "On the Street Where You Live", Musicat easily makes an analogy between measures 1–2 and measures 3–4. The program is helped tremendously by its restriction to make groups that start and end at bar lines. In MusicatW (as with MusicatH), there is no such restriction, and the program is free to make shorter groups; this freedom leads to additional complexity. In this section I first present a run of MusicatW on the first four measures of the melody (for this run and the next I use a metrically-shifted version of the melody — unfortunately, this reverses the implied accents of beats 1 and 3 in each measure, but this is a minor change). The next run and the rest of the runs in this chapter use a later version of MusicatW that can form explicit analogy maps in the Workspace.

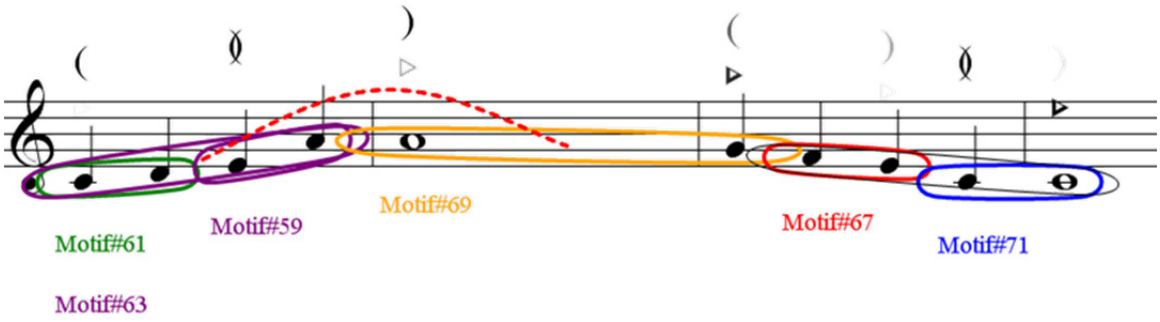

Figure 9.33: "On the Street Where You Live" (run 1).

The main problem with this run of MusicatW is closely related to the problem MusicatH had in its first run on this melody, above (Figure 9.19). In this run, MusicatW has created a yellow group starting on the whole-note A and continuing to the right to include the following quarter-note G. As before, one of the problems is that Musicat hears a strong connection between A and G because they are separated by a single step. It would make more sense for it to hear the stronger connection between the whole-note A and the quarter-note A

coming just before, because they are the same pitch! After all, the program did make the final blue group consisting of a quarter-note C followed by a whole-note C.

The yellow group is ridiculous, but aside from that, the program made reasonable groups, including a meta-group comprised of the two 2-note groups in the first measure. There is also a very weak meta-group comprised of the final two groups in the melody, in measures 3–4. If only the yellow group had not included the G, then other more reasonable groups would have been able to form and the overall grouping structure might have turned out to be understandable as a human-like way of hearing the melody! However, the group structure heard by the program lacks coherence: in this listening performance, only one relationship was heard between groups, and this relationship makes no sense: the meta-group in measure 1 has been linked to the yellow group in measures 2–3. The motif labels are also quite suspect: groups are labeled as "motifs" even though each motif only appears once in this melody: there is only one Motif#61 or Motif#67. Since each group has a different motif number, it is clear that the program has perceived no useful relationships between musical patterns; there is nothing whatsoever here that suggests that the program has heard part of the analogy **(1–2)↔(3–4)** that is essential to this melody.

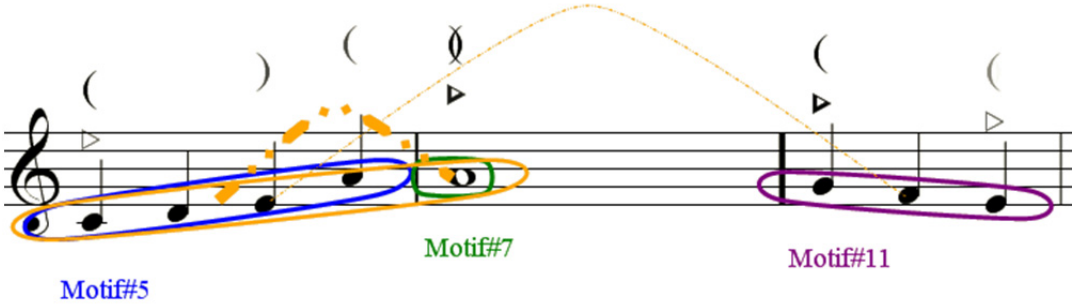

Figure 9.34: "On the Street Where You Live" (run 2, with analogies), mid-run.

In this run, I used a much improved version of MusicatW: analogy maps could be created by the program. Strong analogies are shown as thick yellow arcs with a dash–dot–dot

pattern. Tentative analogies are shows using the same color and pattern, but with very thin arcs. Another important change in this version is that I forced groups to start and end at bar lines, greatly reducing the set of possible groups that the program would consider. I stopped the run before the end of the melody to show how analogies form and grow over the course of a run.

The program has formed entire-measure-long groups in measures 1 and 2 and has combined them into an orange meta-group. The first notes of measure 3 have just been "heard" and grouped together. An analogy has been formed between the first two measures, and surprisingly, it has been perceived to be stronger than the nascent analogy between measures 1 and 3. The first analogy has a high strength because many links (not pictured) have been discovered between the notes of the measure 1 and the whole-note A. Fewer links have been discovered between measures 1 and 3 because measure 3 has just been heard.

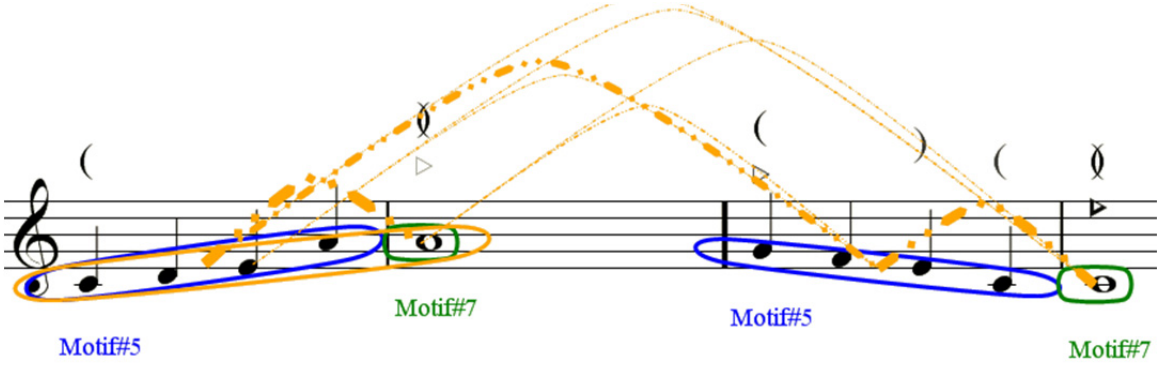

Figure 9.35: "On the Street Where You Live" (run 2, with analogies), end of run.

Figure 9.35 shows the Workspace after all four measures have been heard. Measure 3 has now been perceived as an instance of Motif#5, and the analogy between measures 1 and 3 is now much stronger than it was earlier. The final whole-note C has also been perceived as an instance of Motif#7, just as the whole-note A was earlier. A very weak analogy has been

built between measures 2 and 4; similar weak analogies have been built between several other pairs of measures.

This run, despite some problems described below, demonstrates a tremendous improvement over previous runs of MusicatW and MusicatH, which didn't have this analogy-making ability. As I consider the development of Musicat over the past several years, it is obvious to me that the introduction of these analogy maps was a pivotal moment in improving Musicat's ability to make sense of musical structure. (This version also was aided by the simplifying restriction that groups had to start and end of bar lines, but it seems that this change, which I made several months earlier, was not as important as analogy maps were.) Earlier versions of the program could make links between structures, but analogy maps provided much more. First, they provided a way to collect and coordinate many relationships between the two sides of the analogy. Second, and most crucially, they provided a source of *top-down pressure*: codelets could be posted with the goal of improving an analogy or considering the creation of groups and links suggested by the presence of the analogy. Although earlier versions of Musicat had various top-down pressures, the pressures made possible by the analogy maps provided the leverage, in many runs, for Musicat to stop flailing as it listened and to make progress in creating strong groups and links in the Workspace.

Despite all this, the program still had (and has) a long way to go. This run was quite encouraging, but for a human listener, the connection between measures 2 and 4 would be much stronger — the arc is much too thin for such an obvious relationship — and the tentative analogies between measures 1 and 2 and between measures 3 and 4 would be nonexistent. Also, this run is still missing the crucial larger-scale analogy **(1–2)↔(3–4)**. The program's discovery of the sub-analogy **(1)↔(3)** was the big success of this run (along with a quite reasonable grouping structure).

Musical analogies that appear to us to be trivial (such as mapping the whole-note A onto the whole-note C) can pose huge problems to computer models like Musicat, because *any imaginable mapping* between any two structures might possibly be sensible in the right context. The fact that both of these notes are whole notes might be relevant in this melody, but a similar fact might be completely irrelevant in another melody (*e.g.*, consider a *cantus firmus* bass line composed entirely of whole notes). To distinguish relevant from irrelevant features is extremely hard; as designers of a perceptual architecture, we can't simply tell the program to "ignore pitch" or "ignore rhythm", because either, both, or neither of these rules might apply in a particular context.

Very Simple Melody

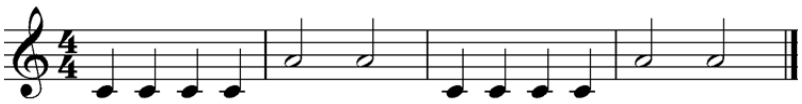

Figure 9.36: A very simple melody.

While working to improve MusicatW and its analogies, we composed a very simple (but not particularly musically pleasing) "melody", which has nearly the same structure (in terms of expected groups and analogies) as the "On the Street Where You Live" melody. We realized it was essential that the program be able to generate the desired analogies in a trivial melody quickly and virtually deterministically (even though the architecture is stochastic). The melody consists of two unrelated measures followed by an exact repetition of those measures. The trivial analogies expected are between measure 1 and measure 3 and between measure 2 and measure 4, because measures 1 and 3 are identical, as are measures 2 and 4. Despite the melody's simplicity, MusicatW still had some problems, but because of this

simplicity it was easier to figure out the sources of these problems. This simple melody served as a useful diagnostic tool.

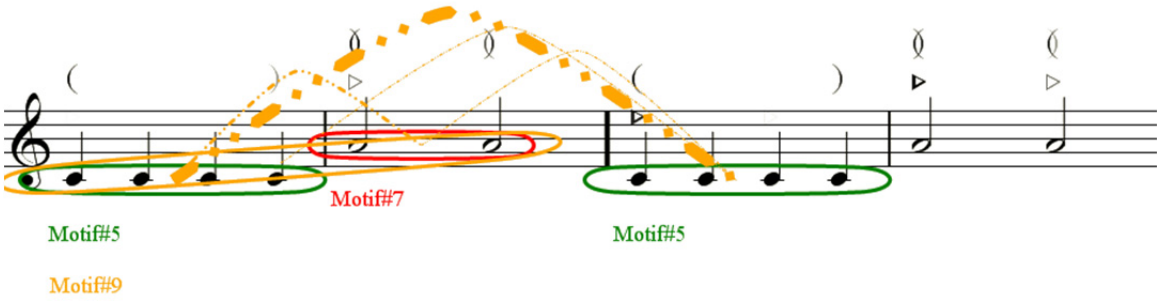

Figure 9.37: Very simple melody, during final processing.

Figure 9.37 shows Musicat after the melody had been presented, but while it was still "listening". Just as in the "On the Street Where You Live" run above (Figure 9.35), an analogy was built between the first two measures, although a human would hear them as unrelated; luckily, Musicat's analogy between these measures was weaker for this melody. Also as in Figure 9.35, each of the first two measures formed a group, and the resulting groups were combined to make a 2-measure meta-group, **(1–2)**. This meta-group is sensible here (even though its constituent measures seem unrelated) because of the temporal proximity of the groups and also because of the thicker bar line between measures 2 and 3 and the exact repetition of measure 1 in measure 3. This repetition has also been recognized by Musicat, as is indicated by the thick analogy arc, representing the strong analogy **(1)↔(3)**.

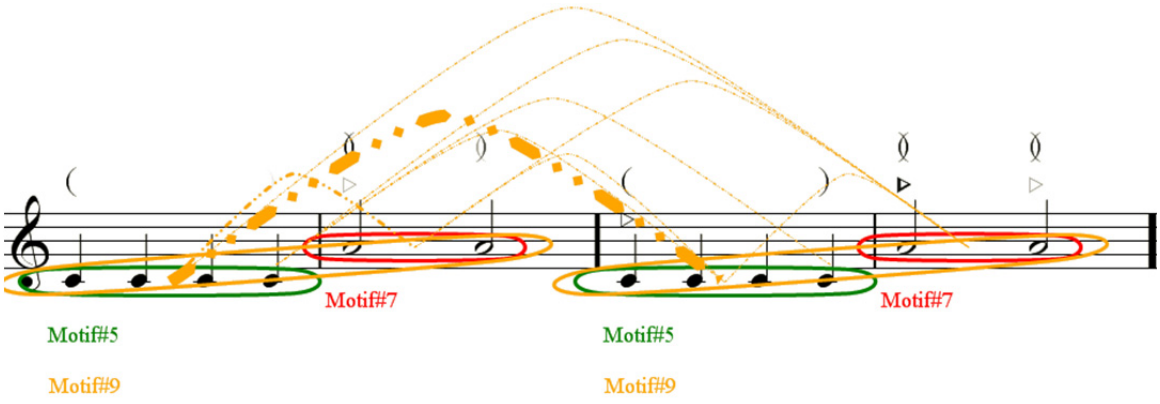

Figure 9.38: Very simple melody, end of run.

The groups and analogies formed by the end of this run are quite similar to those the program formed in the "On the Street Where you Live" run above (Figure 9.35). Each measure forms a group, and measure pairs also form meta-groups: **(1–2)** and **(3–4)**. (Recall that the group **(3–4)** was missing in Figure 9.35.) The analogy **(1)↔(3)** is still strong, and many other tentative analogies have been formed of which the only one that has any significant strength is the analogy **(1)↔(2)**, but it is still very weak, as it should be.

Unfortunately, just as in Figure 9.35, the program has missed the important analogy **(2)↔(4)** — a tentative analogy has been formed, but these are extremely common and so we should only be interested in analogies represented with thicker arcs. Because this analogy was missing, the program also doesn't find the large-scale analogy **(1–2)↔(3–4)**. This tentative analogy is also visible in the figure, but the program has not developed it.

Thus, the simple melody showed us some places where the program should be improved: we needed to ensure that the program would discover the missing analogies. *If* the program were working as desired, the **(2)↔(4)** analogy and the **(1–2)↔(3–4)** meta-analogy could be discovered in the following way: after the discovery of the **(1)↔(3)** analogy, a top-

down codelet might notice that the analogy crosses a relatively thick bar line (the bar line between measures 2 and 3). Then, this codelet would post other codelets to the Coderack to search for a *parallel* analogy between measures 2 and 4. Long-distance links would be built between notes in these two measures, and thanks to them the parallel analogy would easily be seen.

I added codelets to the program to make possible the scenario just described, but even so, on almost all runs, MusicatW failed to see the desired meta-analogy, **(1–2)↔(3–4)**. At this point in MusicatW's development, the program was quite complex, and I believe it simply spent too much time exploring other possibilities, and the top-down pressure afforded by the program's analogies was still not strong enough to overcome its tendency to "flail" too much. My experience with this simple melody, along with the program's trouble with "Sur le pont d'Avignon" (next section), soon led to a significant rewrite of the program.

Sur le pont d'Avignon

Recall the 8-measure melody of "Sur le pont d'Avignon" from Chapter 6:

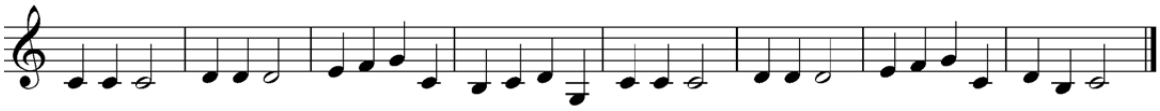

Figure 9.39: Sur le pont d'Avignon.

The current version of Musicat can readily form several layers of meta-groups and make analogies of various sizes, including **(1–2)↔(5–6)**, **(3–4)↔(7–8)**, as well as the meta-analogy **(1–4)↔(5–8)**, mapping the entire first half of the melody onto the second half:

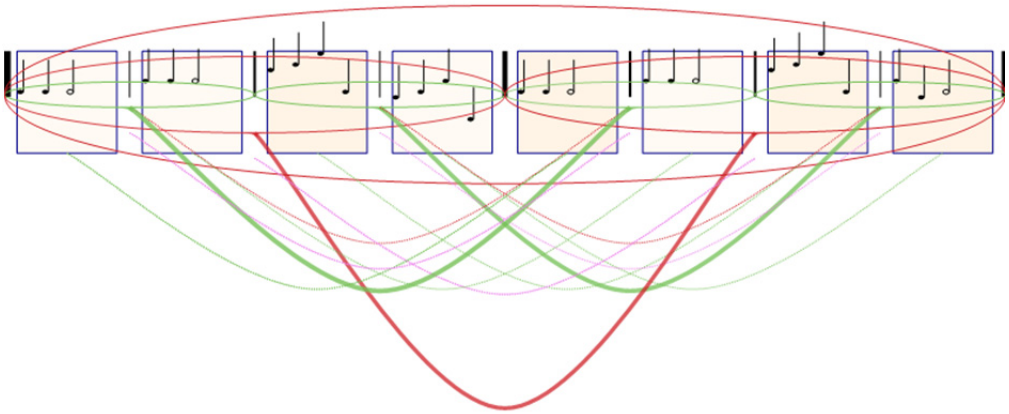

Figure 9.40: Sur le pont d'Avignon, result of a run using the latest Musicat (repeated from Chapter 6).

But now consider the results from a run using MusicatW. The difference is striking:

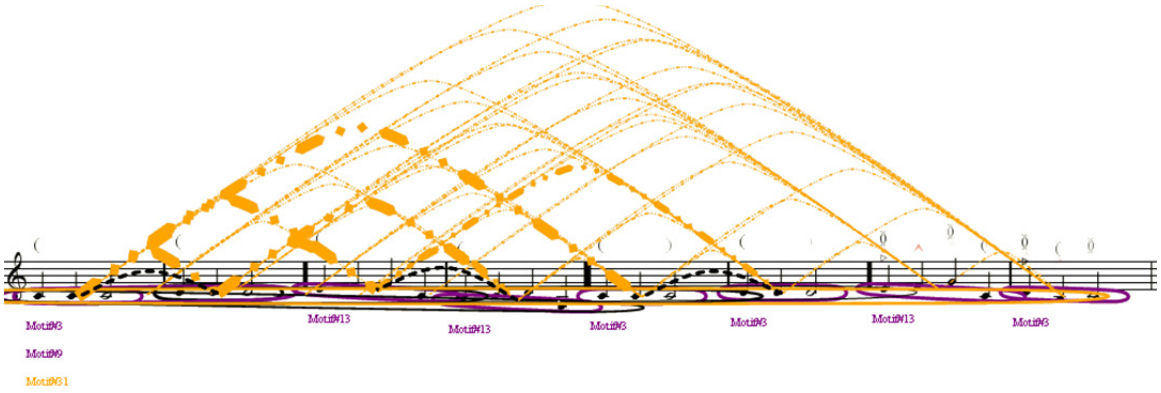

Figure 9.41: Far too many analogies in "Sur le pont d'Avignon" — an embarrassment of riches.

Clearly, this version of the program has completely failed to hear the structure of the melody. Instead, it has created a ridiculous number of tentative analogies: one has been formed for every single possible pair of groups! The strong analogies that have been formed seem to favor the early part of the melody, and the ending of the melody is not part of *any* strong analogy. This stands in contrast to the run of the latest version of Musicat (Figure 9.40), in which the first and second halves of the melody are placed on equal footing, as is shown by the large analogy connecting both halves.

This failure is important because the embarrassing abundance of tentative analogies that Musicat considered during the run vividly illustrates the difficult problem the program faces. In general, there are so many potential analogies between entities in the world that the combinatorial explosion is overwhelming unless one has heuristics to determine which analogies might be most pertinent. In Figure 9.41, MusicatW is looking at so many possible mappings that it simply flails, failing to perceive the most crucial features of the musical structure. The capacity to keep the focus on *what matters*, and to extract the essence — the *gist* — of a situation, is extraordinarily difficult, especially in real time. Musicat needs this ability, but it is obvious that, in this run, MusicatW was lacking that ability.

After I saw this particular "listening performance" in the spring of 2011, I was quite frustrated by the state of the program. In the fall of 2001, I therefore decided to make a radically simplified version of Musicat that would focus its attention on what I deemed to be the most important aspects of melody perception: making reasonable groups and making larger analogies based on them, and also inspiring new groups.

BinaryCat

The early versions of Musicat included many codelets that examined the features of individual notes (note perspects) and made links between notes. Because of the run on "Sur le pont d'Avignon" in which the program failed to make any large-scale analogies (*e.g.*, mapping the first four measures onto the last four measures), I decided to try an experiment in which I removed practically all information about individual notes from the melodies given to the program. The program already had been restricted to making groups that were at least one measure long, but it still spent a lot of time "listening" to individual notes. I realized that if the program considered the music at a higher level of granularity, the patterns it had failed to perceive before should suddenly become very obvious. For example, if one

thinks of the "Sur le pont d'Avignon" melody in terms of measure-sized chunks and assigns a new letter such as **A** or **B** to each measure that has different notes from a previous measure, the melody is converted from this…

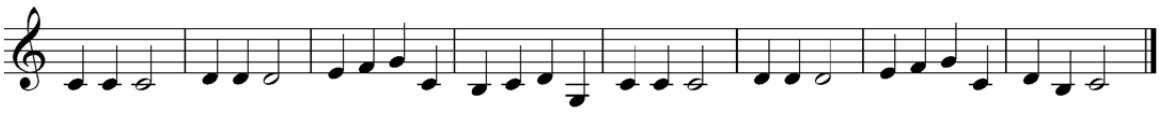

Figure 9.42: "Sur le pont d'Avignon".

…to this:

A B C D A B C E

Surely the large-scale analogy represented in terms of these symbols, **(ABCD)↔(ABCE)**, is simpler to perceive than the same analogy based on individual rhythms and notes. The melody is given to Musicat in the following form, which looks considerably more complex:

QC QC HC | QD QD HD | QE QF QG QC | QB3 QC4 QD QG3 | QC4 QC HC | QD QD HD | QE QF QG QC | QD QB3 HC4

Of course, even MusicatH had the ability to make groups, and these groups should have been able to provide the chunking necessary to convert this note-level representation into a chunked representation such as "**A B C D A B C E**". However the program seemed to be spending too much time with individual notes. I wanted to "hide" individual notes from Musicat temporarily while working on the codelets that could make analogies such as **(ABCD)↔(ABCE)**. I wanted to start with a blank slate and make a new type of Workspace and a new set of codelets that were aimed squarely at making larger analogies (and I planned to take lessons learned from this experiment and apply them to MusicatW).

In my first attempt to implement this idea of hiding the internal structure of each measure in order to focus the program's attention on larger analogies, I had the (naïve) inclination to simply assign a *random* bit vector to each measure with unique notes, and I made a new program called "BinaryCat" (so-named because it used vectors of 0's and 1's). This idea came to me because I had been reading about Pentti Kanerva's memory model, Sparse Distributed Memory (Kanerva, 1988), which makes elegant use of random high-dimensional binary vectors. But in this particular experiment, the use of random bit vectors was not well motivated (why not drop all notes and just use the letters **A**, **B**, **C**, and so on to represent measures?). Therefore, I worked on BinaryCat only for a few days before deciding that I should include a little more information about each measure so that the program could make useful analogies. For example, in the example in the previous paragraph, the measure labeled **D** (measure 4) looks completely different to the program than the measure labeled **E** (measure 8), but these two measures are actually quite similar.

What minimal information should be used to represent each measure without returning to a full-fledged description of every note? Douglas Hofstadter and I had already discussed how rhythmic similarity seemed more primal than pitch relationships in the initial stages of analogy formation, so I decided to give some rhythmic knowledge of the melody to this experimental program. I replaced the random bit vectors of BinaryCat with vectors representing note attack times, leading to the program RhythmCat.

RhythmCat

RhythmCat is a version of Musicat that has no knowledge of pitch: input to the program is simply a list of note attack times (RhythmCat doesn't even understand rests — it just hears when notes begin). In other words, RhythmCat listens to a "flattened" version of melodies: imagine that for each melody, RhythmCat takes the original sheet music notation

and slides all the notes up or down onto one staff line. For example, the "Sur le pont d'Avignon" melody turns into the following non-pitched, percussive sequence:

Figure 9.43: "Sur le pont d'Avignon", rhythm only.

Our decision to make a rhythm-only program may seem strange, but experiments have suggested that people are better at recognizing a melody based on its rhythm than on its pitch contour — see, for example, (Jones, 1993). If rhythmic patterns are indeed more fundamental than pitch patterns in terms of melody recognition, then it is not a large leap to suppose that rhythm is similarly fundamental for the analogy-making that the program needs to model. Pitch is also very important, but ignoring it temporarily while developing RhythmCat made it easier to focus my efforts (and the program's efforts) on larger-scale analogy making.

BinaryCat's code was the basis of RhythmCat. BinaryCat, however, was essentially a new program, although it utilizes small parts of MusicatW's code, such as the code that implemented the Coderack and the main program loop for simulating the arrival of new notes and running the right number of codelets per note. I tossed out the old user interface, the old Workspace, all the old codelets, and so forth, and started anew. I created new codelets, restricted the program so that it was required to group together *measures* (not notes), a new analogy data type, new Workspace code, and so on.

The following figure shows an early run of RhythmCat on a simple rhythm.

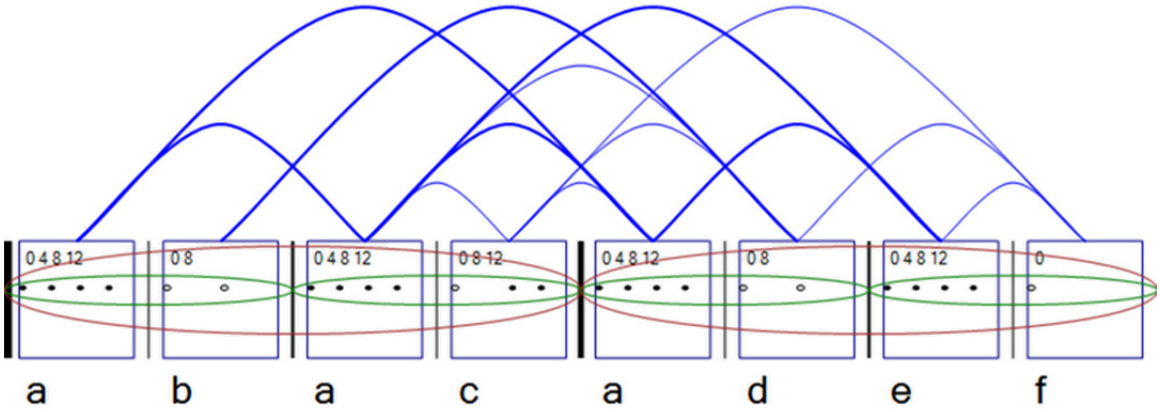

Figure 9.44: A run of an early version of RhythmCat on a simple rhythm.

This run is from a version of the program that included measure links (in blue) but didn't yet have the ability to make analogy maps. However, it did have the ability to assign labels such as "**a**", "**b**", or "**c**" to measures. For example, most of the instances of the rhythm QQQQ in the figure above are labeled with "**a**" (the final instance of this rhythm is labeled as "**e**" because the program has not yet noticed this connection. Later in its development, RhythmCat gained the ability to use the prime symbol (☒) to mark measures as variants of other, earlier measures. The next figure shows a run from such a version, which also includes a new analogy-making ability. Also, codelet activity is indicated here by a series of yellow dots above measures where activity was taking place.

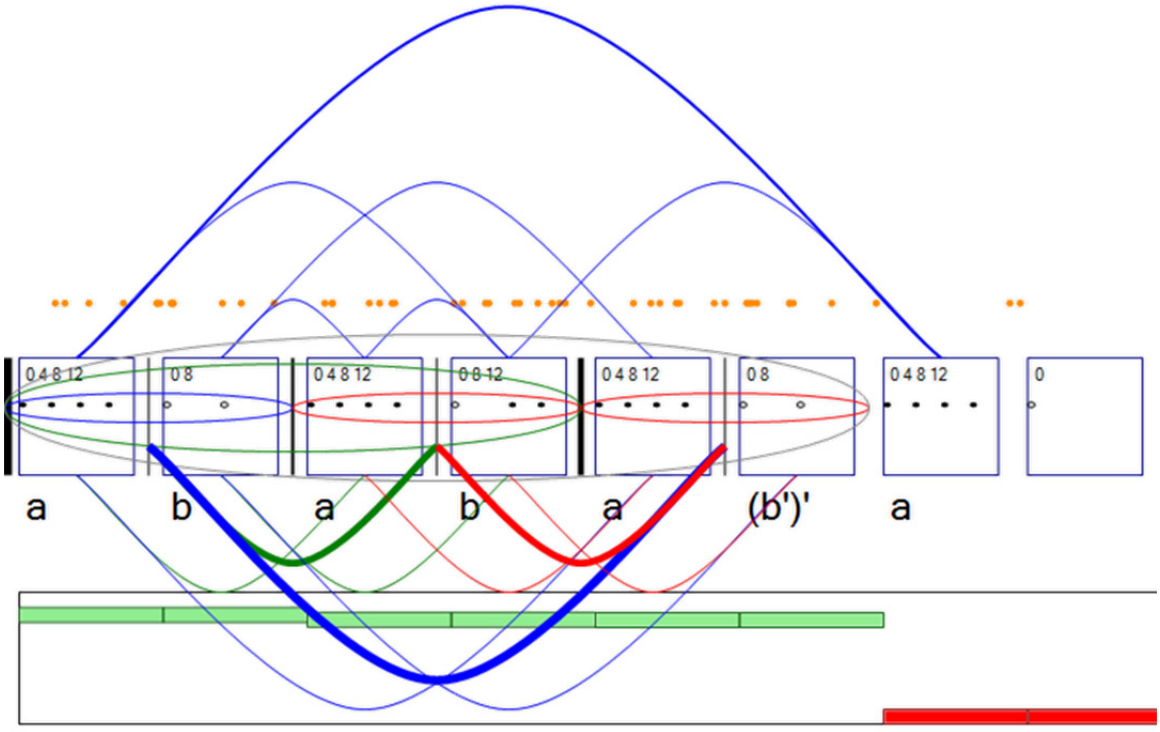

Figure 9.45: A run of a later version of RhythmCat, whose display resembles that of the latest version of Musicat.

Figure 9.45 should remind the reader of the figures in Chapters 5–7, where are based on the most recent version of Musicat. (Notice a curious thing in this run: measure 2 has been labeled as "**b**". Measure 4 has been heard as a variant of measure 2, and is labeled "**b′**", but the "prime" symbol, unfortunately, has been covered up by a red analogy arc. Measure 6 has the same rhythm as measure 2, but it has been heard as a variant of measure 4, so it is labeled "**(b′)′**".)

Because it was the result of a substantial rewrite of the program code, RhythmCat has many differences from earlier versions of Musicat (although the basic FARG architecture is still the foundation of the program). The following list describes the major features of RhythmCat that distinguish it from MusicatW.

- Measures consist only of a list attack times, not pitched notes.
- There is no Slipnet.
- There is no Motivic Memory. Groups are no longer labeled with motif numbers.
- Note links do not exist.
- Rhythmic links between entire *measures* are the most elementary similarity structure in the Workspace.
- The measure (not the note) is the smallest-sized structure for the purpose of group and analogy formation.
- Group scoring uses a simple weighted linear combination (the strategy of running group score statistics was removed).
- Bar-line thicknesses are determined by codelets.
- Measure links and analogies can create special "Relationship" objects.
- Analogies always form a link between two *groups*.
- Analogies are a collection of these relationship objects in the Workspace.
- Analogies use a new scoring formula.
- Many new high-level codelets exist that add more top-down pressure.
- Temperature is computed *locally*, for a single measure or a small range of measures; there is no global temperature.
- High-temperature (low-happiness) areas get more attention from codelets.
- I implemented the add-in for Visual Studio that provides "instant" feedback after code compilation.

It may be obvious from the list above that many of these features of the program are also part of the latest version of Musicat. This is because RhythmCat rapidly evolved into the

latest version of Musicat — I did not return to the MusicatW code base. After working for several months on RhythmCat, we had seen progress but it was clear that we would have to come back to using pitch information, because there were examples in which pitch information would have helped to make certain analogies very obvious. For instance, in "Twinkle, Twinkle, Little Star", all odd-numbered measures have the rhythm QQQQ, while all even-numbered measures have the rhythm QQH. RhythmCat notices all these repetitions of the same rhythms, but it can't find any consistent and interesting higher-level structure without pitch. Pitch is crucial to the sense of the structure of this melody.

Because RhythmCat showed much more promise than MusicatW did in terms of making larger-scale groups and analogies, we decided to add pitch to RhythmCat, which resulted in the latest version of Musicat (that is, the version described in the rest of this thesis). In this new version of the program (which I will simply call "Musicat"), rhythm is the primary musical feature used to make analogies. Pitch was added in carefully during development, so that it would provide extra information to the program in cases like "Twinkle, Twinkle, Little Star" without overwhelming the strong clues afforded to the program by its focus on rhythm.

I have already described Musicat, but to complete the story of its evolution I will list the major new features that were added to RhythmCat in order to arrive at the current version of Musicat. The new Musicat includes the following significant new elements, among others:

- Pitch-based relationships, including contour relationships and tonal relationships;
- Inferred pitch alphabets;

- Support for upbeats (pickup notes);
- The ability to store and resurrect large and complex structures;
- A graphic display of measure-by-measure happiness values;
- Group and analogy expectations;
- A new user interface focused on showing strong groups and analogies; and
- A slider in the user interface providing a way to vary the displayed detail level.

This concludes the history of Musicat up to the current version. It is quite clear that the latest version of Musicat is a huge step forward compared with the flailing versions that came before, especially considering melodies such as "Sur le pont d'Avignon". Many of the ideas of the earlier versions, however, may prove useful in future versions of the program. The next chapter will discuss the major lessons learned during the development of Musicat, will review the current state of the program, and will suggest priorities for the next steps in Musicat's continuing evolution.