THE *PERLHUMDRUM* AND *PERLLILYPOND* TOOLKITS FOR SYMBOLIC MUSIC INFORMATION RETRIEVAL

Ian Knopke
Goldsmiths Digital Studios
ian.knopke@gmail.com

ABSTRACT

PerlHumdrum is an alternative toolkit for working with large numbers of Humdrum scores. While based on the original Humdrum toolkit, it is a completely new, self-contained implementation that can serve as a replacement, and may be a better choice for some computing systems. *PerlHumdrum* is fully object-oriented, is designed to easily facilitate analysis and processing of multiple humdrum files, and to answer common musicological questions across entire sets, collections of music, or even the entire output of single or multiple composers. Several extended capabilities that are not available in the original toolkit are also provided, such as translation of MIDI scores to Humdrum, provisions for constructing graphs, a graphical user interface for non-programmers, and the ability to generate complete scores or partial musical examples as standard musical notation using *PerlLilypond*. These tools are intended primarily for use by music theorists, computational musicologists, and Music Information Retrieval (MIR) researchers.

1 INTRODUCTION

Humdrum [5, 3, 4] is a computer-based system pioneered by David Huron for manipulating and querying symbolic representations of music. Unlike notation-based systems such as GUIDO or MusicXML, Humdrum is primarily intended as a set of analytical tools to aid music theorists, musicologists, acousticians, cognitive scientists, and MIR researchers, among others. Also, Humdrum data files are differentiated from stored-performance formats such as MIDI, in that almost all have been encoded by hand from musical scores; more than 40,000 Humdrum files have been encoded to date, the majority of which are freely available online [1, 6].

The original motivation for this research was the desire to use the Humdrum resources in a series of music analysis projects, coupled with a certain frustration with the lack of functionality of some of the original tools which were reliant on older Unix environments and were difficult to make work under current versions of Linux. Other Humdrum tools, such as those for entering MIDI information, are associated with hardware that currently unavailable in most operating systems. Also, many of the problems that the project explored required writing additional programs that worked directly with the Humdrum data, such as data collection and extraction across large numbers of scores, and it was simply easier to write them fresh in Perl than to try to accomplish the same thing based on one of the older Awk tools. Over time a collection of alternate tools, extensions, and replacements began to take shape, and at some point it simply made sense to collect everything under a single code base that shared methods for common tasks in a object oriented framework. In the process, it became possible to rethink some aspects of the original Humdrum Toolkit, and introduce some new possibilities that are useful when working on these sorts of problems.

Additionally, from working with various musicologists, it became clear that a system was needed for automatically producing musical excerpts in common musical notation, as a method for displaying search results. This led to the creation of the *PerlLilypond* programs, that provide a scriptable environment for generating notation. The two systems are designed to complement one another.

The author refers to the two systems as *PerlHumdrum* and *PerlLilypond*. This is primarily to differentiate their names the original programs. However, within the Perl environment, the module names `Humdrum` and `Lilypond` are used. This is partially because of traditional naming conventions in Perl, but also because the author doesn't see the necessity of repeatedly inflicting an extra syllable on developers and users during tasks such as multiple object creation.

2 OVERVIEW OF *HUMDRUM*

The overall *Humdrum* system is comprised of two main parts: a set of file-based storage formats for time-aligned symbolic data, and the accompanying collection of UNIX-based tools for manipulating and querying this data. The *PerlHumdrum* system discussed in this paper is designed to simplify operations on the former, while providing an alternative to the latter.

Humdrum data files consist of symbols representing musical or other symbolic information, arranged in time-aligned columns known as *spines*. As an example, the first measures

of a Bach chorale and the representative Humdrum excerpt are shown in Figure 1 and Table 1 respectively.

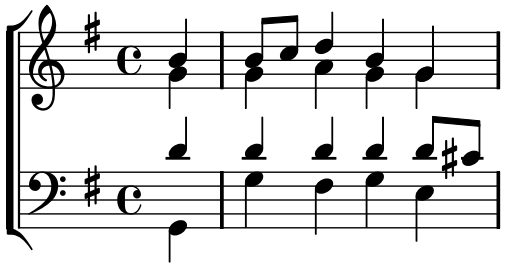

Figure 1. *Uns ist ein Kindlein heut' gebor'n*, J. S. Bach

**kern	**kern	**kern	**kern
*bass	*tenor	*alto	*soprn
*k[f#]	*k[f#]	*k[f#]	*k[f#]
4GG	4d	4g	4b#
=1	=1	=1	=1
4G	4d	4g	8b#
.	.	.	8cc
4F#	4d	4a	4dd
4G	4d	4g	4b
4E	8d	4g	4g
.	8c#	.	.

Table 1. A Humdrum `Kern` encoding of Figure 1

Humdrum files consist only of ASCII characters, including tabs and spaces, and are easy to create, view and edit in any standard text editor. Each vertical spine represents a single stave of music from the original score. Where new events in the original score occur horizontally, moving from left to right. In the Humdrum version, new events are added to the bottom of each spine. In many ways, Humdrum encodings can be seen as normal scores rotated clockwise by 90 degrees. Items aligned horizontally occur at the same time, and a single horizontal line of data is known as a *record*. Items marked with asterisks (*,**) are metadata events or *interpretations* that give instructions as to how data spines are to behave. For instance, spines can be added, subtracted, or exchanged, among other operations. Interpretations are also used to indicate key and time signature.

The first item in each column indicates the type of Humdrum format in use. In this case the standard `Kern` format that is more or less analogous to common-practice music notation is being employed, with notes encoded as a combination of numeric value and letter, representing the rhythmic and pitch values. Measure numbers are preceded by equal signs and are repeated in all columns. The letter c indicates middle *c* (261.63 Hz), with lower octaves being represented by upper-case letters, higher octaves by lower-case letters, and larger distances from middle *c* by accumulated lettering. A set of *c* notes going from lower to higher octaves would be encoded as "CCC, CC, C, c, cc, ccc".

While the `Kern` format is the most common type of Humdrum encoding, many other types of encodings exist, including frequency, scale degrees, melodic and harmonic intervals, solfege symbols, set-theory representations, and even psychoacoustical representations such as barks or critical band rates. Spines may be added or removed from a file as necessary, or manipulated in various other ways. Multiple encoding schemes can also be used in a single file, and the user can define their own types to accommodate specific needs.

Coupled with data storage part of Humdrum is the toolkit, a set of programs for working with *Humdrum* data files. The programs encompass a rich set of functionality, including devices for manipulating and querying, and summarizing this kind of data. It is difficulties with these tools in particular that *PerlHumdrum* is designed to address.

3 *PERLHUMDRUM* USAGE

The original Humdrum toolkit operates using the standard UNIX command line paradigm. Commands in Humdrum generally consist of Awk scripts that take command line options and an input Humdrum file, and produce another Humdrum file. Multiple commands can be strung together using pipes, and intermediate results can be stored as temporary files to undergo further processing. For instance, one of the simplest of all Humdrum programs is the `census` command, used here to obtain a list of overall statistics for a kern file called *bach.krn*:

```
census -k bach.krn
```

This produces a text file containing various statistics about the original piece, such as the number of notes, number of comments, and other basic information.

In *PerlHumdrum*, the basic ingredient is the `Humdrum` object. The equivalent program to the above using *PerlHumdrum* is shown below.

```
use Humdrum;

my $c=Humdrum->new('bach.krn');
my $dat=$c->census(k=>1);
```

Command line options from the original programs are available as parameters to the *PerlHumdrum* methods. Great care has been taken to ensure the best possible compatibility between the original toolkit and this one; most of the original programs have been converted to methods, and great

care has been taken to ensure that all options have been accounted for and behave identically.

Applying this command to multiple files, to get a sum of notes for instance, is almost as simple:

```
use Humdrum;

my $sum=0;

foreach my $file(@filelist){
    my $c=Humdrum->new($file);
    my $dat=$c->census(k=>1);
    $sum+=$dat->get_datatokens();
```

In the original toolkit, processes could be chained together by saving the results of each stage as a temporary file and using it as input to the next, or by using the pipe symbol. The example below demonstrates a short processing chain that removes the non-note elements from the `kern` file "bach.krn" and then performs a similar census operation.

```
kern -x  bach.krn | census -k
```

In *PerlHumdrum*, under normal circumstances, a `Humdrum` object is considered immutable; that is, the output of one operation produces another `Humdrum` object.

The previous set of commands can be translated into a *PerlHumdrum* program as follows:

```
use Humdrum;

my $c=Humdrum->new('bach.krn');
my $k=$c->kern(x=>1);
my $dat=$k->census(k=>1);
```

Here is an even more succinct version:

```
use Humdrum;

my $c=Humdrum->new('bach.krn');
my $dat=$c->kern(x=>1)->census(k=>1);
```

Obviously, much more complex examples are possible.

4 ADVANTAGES

The *PerlHumdrum* toolkit has several advantages over the original Humdrum toolkit.

The primary advantage is that the object oriented paradigm makes it simple to apply complex data analysis and modification operations to multiple files, through the use of loops or other control structures. While this is possible in the original command-line syntax using command-line bash shell loops and judicious use of piping, the results are often brittle and not well-suited to large-scale analyses of symbolic data as is often required in MIR research. We believe that these limitations of the current toolkit have limited the uses of existing `Kern` repositories in many MIR research areas.

A second, perhaps less-obvious advantage is that the original Humdrum toolkit makes complete sequential passes through entire Humdrum files and then passes that data on to further commands through pipes. While many operations are possible in this way, others are extremely difficult. Consider the situation of trying to analyze all of the cadences in a piece or set of pieces. Cadences are most easily-identified by working backwards through a file, finding the resolution in reverse (I-V) and then working back until the "start" of the cadence is detected. Doing such work using only forward sequential passes, perhaps with an additional operation once the cadence is located, is extremely difficult. *PerlHumdrum* provides several different methods of proceeding through a Humdrum file, including reverse iteration and ranged iteration. The array of objects is also available to the user, making it possible to define any kind of non-sequential operation the user might like (only even tokens, for instance).

Other advantages are:

- consistent use of OOP principles, instead of "imperative" Awk-style scripts simplifies the addition of new commands.
- The use of perl provides a common base for other "data-munging" operations that Perl is commonly used for. This makes it easy to connect Humdrum operations to databases, linear algebra libraries, or even CGI scripts for web pages, to give just a few examples.
- Unlike the Awk and C basis of the original toolkit, *PerlHumdrum* runs natively on many different computing platforms, including Unix, Macintosh, and Windows systems without recompiling.

5 *PERLLILYPOND*

A parallel package, *PerlLilypond*, has been developed that can take information from a Humdrum object and convert it into common practice music notation. As the name suggests, *PerlLilypond* uses *LilyPond* [14] as the underlying notation engine.

PerlLilypond has two primary virtues. First, *PerlLilypond* can be easily scripted and called within other programs. While *PerlLilypond* can be used to generate entire scores, and can function as an alternative, non-proprietary notation system. The intended use is for displaying multiple excerpts from a collection of Humdrum scores, such as all of the cadences across a corpus, without individual human intervention and editing. This is extremely difficult to do with any of the more-familiar commercial notation programs.

Secondly, and perhaps more importantly, *PerlLilypond* displays the results of symbolic searches in a form that is accessible to most musicians. Experience has shown that many traditional musicologists and theorists without a background in MIR have difficulties with non-traditional notation formats. Displaying results in a traditional notation format makes it much easier for many music researchers to use this technology, and we feel that the lack of such a system has probably been the single biggest impedement to the adoption of these technologies within that community. Also, MIR researchers themselves, with traditional music training, may find it much easier and more succinct to display results in this format.

There are two ways to use *PerlLilypond*. The easiest is simply to pass it a *PerlHumdrum* object, which is the common output of most *Humdrum* operations, and *PerlLilypond* will do all the work of converting the object into its own internal format. Thus, the entire process of printing a Humdrum score or any Humdrum data can be reduced to the procedure outlined below.

```
use Humdrum;
use LilyPond;

my $humdrumexcerpt=Humdrum->new('Bach.krn');
my $l=Lilypond->new($humdrumexcerpt);
$l->print();
```

A second way is to gradually build up an object by adding notes, staves, and other musical structures sequentially. In this mode, a *PerlLilypond* object can be considered a kind of long pipe that we keep pushing objects into until our excerpt is finished. Other "container" objects, such as chords or tuplets, can be created as needed and added to a stave. No facility is provided for editing materials inside a *PerlLilypond* object once they have been allocated, although references to the underlying raw arrays is available for end users who wish to build their own subroutines. This is because such editing tools are already in *PerlHumdrum*.

The procedure for creating a short excerpt consisting of a note, a chord, and a final note is shown below:

```
my $ll=Lilypond->new();
$ll->addstave('soprano');
$ll->addnote('soprano',{step=>'d'});

my $lh=Lilypond::Chord->new;
$lh->add(Lilypond::Note->new(
    {duration=>8}));
$lh->add(Lilypond::Note->new(
    {step=>'e',duration=>8}));
$lh->add(Lilypond::Note->new(
    {step=>'g',duration=>8}));
$ll->addchord('soprano',$lh);
$ll->addnote('soprano',
    {step=>'g',duration=>4,dots=>1});
$ll->makeexcerpt();
```

First, a new `Lilypond` object is created and a stave named *soprano* is added. Following the addition of the first note, a `Lilypond::Chord` object is created and three notes are "poured" into it in sequence. The resulting object is then added to the stave, followed by a final note. Finally, the `makexcerpt` method is called, calling the *Lilypond* binary, and generating the excerpt in all output formats.

Once a user is finished constructing a *PerlLilypond* object, various commands are available to actually get output as PNG or postscript output, transparently created by calling the actual *LilyPond* binary.

Facilities are also provided for displaying large numbers of excerpts as PNG graphics on an HTML page. HTML was chosen as the primary output format for excerpts because web pages are essentially unlimited in length, can easily align images, and can be easily viewed in practically any computing environment. An example of some output generated directly from a Humdrum file is shown in Figure 2. Here, a cadence has been automatically detected and marked with an 'X'.

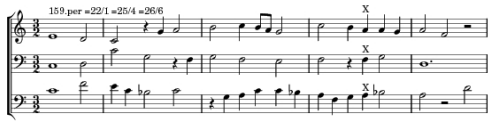

Figure 2. A simple example of web page output

One especially useful facility of *PerlLilypond* is the ability to introduce "hidden" elements that are not seen, but instead act as spacers between other notes. This is extremely useful for displaying the results of aligned sequences of notes, such as those that might be produced by musical applications of various alignment algorithms [17, 9]. At present, *PerlLilypond* captures about 60 percent of the capabilities of *LilyPond*, including almost everything connected to common notation. Development is ongoing to add the remaining alternate notation styles such as percussion notation.

6 EXTENSIONS

PerlHumdrum also provides a number of additional capabilities that are not present in the regular Humdrum toolkit.

6.1 Support for Analyses of Multiple Scores

With the exception of the `patt` and `grep` tools, Humdrum is cumbersome for the analysis of large scores. Such procedures, in the original toolkit, often involves some tricky

shell scripting, writing the results to a log file, and then providing an additional program to process these separately. This is especially difficult if the cumulative results need to be recorded, or if the results of each analysis needed to be inserted into a database (for instance). *PerlHumdrum* provides several "meta-tools" that make it simple to apply a referenced procedure or set of procedures to a collection of files. Figure 3 shows the final results of one such operation, in which the number of common rule violations in the original Bach Chorales have been accumulated as percentages of files of the original collection ([2, 10]).

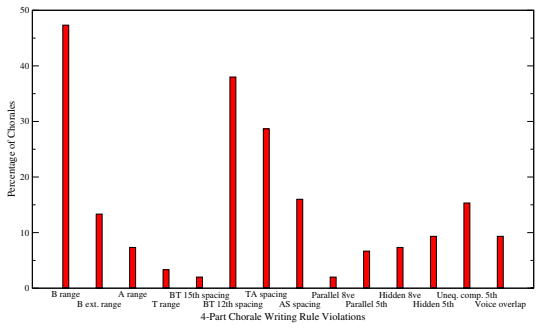

Figure 3. Common rule violations in Bach Chorales

6.2 Graphing Capabilities

In a similar vein as the above, a set of procedures has been provided to facilitate easy graphing of extracted results. The most common variety of this is the "scatterplot" diagram, such as that shown in Figure 4. This provides an easy way to visualize aspects of entire collections, and such graphs are of great interest to musicologists ([7, 8, 18]).

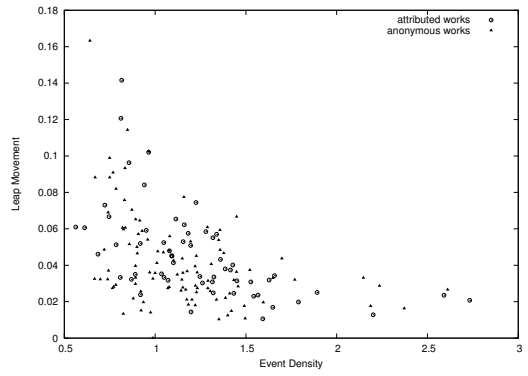

Figure 4. Comparing an entire collection

6.3 MIDI File Conversion

The original toolkit has the ability to convert Humdrum files to MIDI format. There is also a limited facility for converting direct MIDI input from a MIDI-in port into Humdrum, but this relies on a specific hardware library and only functions under Microsoft DOS. *PerlHumdrum* provides a `MakeMidi` method that will convert a MIDI file into Humdrum notation. `MakeMidi` is not currently very good at making assumptions, and relies heavily on the MIDI files being well-behaved To get good results, MIDI files must include tempo and key signature, isolate each stave to a different track, and should probably be heavily quantized. It currently does not handle more complicated tuplets such as quintuplets. However, the current capabilities work well enough for most common music notation scores, and additional improvements are being implemented.

6.4 Integration with Other MIR Systems

While not technically an aspect of *PerlHumdrum*, one very unique aspect of Perl is the *Inline* facility, which allows programmers to actually include foreign code directly inside a Perl source code file. This makes it possible to mix C, Java, Octave, and many other languages directly into Perl. The MIR field currently has many different disparate toolkits designed to handle specific problems, but very few complete systems. Nor do such systems seem likely in the future, as writing programs in multiple languages is usually extremely complex. Using Inline and Perl, it is possible to construct larger programs combined of multiple languages and source code files, including *PerlHumdrum*. We have used this facility in the past, and found it extremely effective, yet it still remains unknown in the MIR community as a whole.

7 FUTURE WORK

Most of the functionality of the original toolkit has been implemented in *PerlHumdrum*. However, a few of the more obscure options of some of the original toolkit programs are currently unsupported, have undergone changes in function in this new context, or have unfortunately been recently demonstrated as somewhat less than robust. Regression testing has helped considerably with these situations, and the entire toolkit is approaching something close to a real, distributable replacement for the original.

It is intended that *PerlHumdrum* be made freely available as is the original Humdrum toolkit, probably under the Gnu Public License [13]. The standard method for making code available to the Perl community is by distributing the modules through the CPAN network [15]. This makes it extremely easy for Perl users to download and install the entire set of modules from within Perl itself. This conversion has yet to be completed; however, current versions of *Perl-*

Humdrum are available through direct email contact with the author.

Several other extensions to *PerlHumdrum* are currently planned or in progress, including: conversion classes to other notation formats such as GUIDO, connections to databases such as PostGreSQL [16] for large-scale storage of music fragments, better integration with the Perl MIDI classes, and the ability to synthesize and play fragments directly using Timidity.

The *PerlLilypond* notational facility is still under development and has difficulty with some advanced notational situations. It has not been well-tested on many types of contemporary music, primarily because there are fewer examples of this style of music in the various *Humdrum* repositories.

The object-oriented nature of *PerlHumdrum* makes it natural to consider building a graphical interface for Humdrum. This has been attempted before [12, 11, 19], but the results have never been satisfactory. Development of such a resource would also be extremely helpful in aiding the adoption of these tools by musicologists and music theorists.

Finally, the set of operations available in Humdrum are probably the most comprehensive set of symbolic music operations in the MIR world. We would like to adapt these tools to operate on other possible file formats. The best candidate at this time is the MusicXML format. In fact a rudimentary version of this already exists, and will be completed as time and necessity permit.

8 ACKNOWLEDGMENTS

The author wishes to express his sincere thanks to Frauke Jürgensen for her ongoing and enthusiastic support of this project; her advice and patience have been invaluable. Additionally, I would like to acknowledge the encouragement of both Craig Stuart Sapp and Don Byrd in these efforts.

9 REFERENCES

[1] CCARH. Humdrum virtual scores, 2006. `http://kern.ccarh.org/`.

[2] B. Gingras and I. Knopke. Evaluation of voice-leading and harmonic rules of j.s. bach's chorales. In *Proceedings of the Conference on Interdisciplinary Musicology*, pages 60–1, March 2005.

[3] D. Huron. The Humdrum Toolkit: Software for music researchers. Software package, 1993.

[4] D. Huron. *Humdrum* and *Kern*: selective feature encoding. In E. Selfridge-Field, editor, *Beyond MIDI: The Handbook of Musical Codes*, pages 375–401. MIT Press, Cambridge, MA, 1997.

[5] D. Huron. Music information processing using the Humdrum Toolkit: Concepts, examples, and lessons. *Computer Music Journal*, 26(1):15–30, 2002.

[6] David Huron. Humdrum scores, 2006. `http://www.music-cog.ohio-state.edu/Humdrum/`.

[7] F. Jürgensen and I. Knopke. A comparison of automated methods for the analysis of style in fifteenth-century song intabulations. In *Proceedings of the Conference on Interdisciplinary Musicology*, page n.p., 2004.

[8] F. Jürgensen and I. Knopke. Automated phrase parsing and structure analysis in fifteenth-century song intabulations. In *Proceedings of the Conference on Interdisciplinary Musicology*, pages 69–70, March 2005.

[9] J. Kilian and H. Hoos. MusicBLAST - Gapped sequence alignment for MIR. In *Proceedings of the International Conference on Music Information Retrieval*, pages 38–41, October 2004.

[10] I. Knopke and B. Gingras. New approaches for the analysis and teaching of chorale harmonizations. Presentation at the Royal Conservatory of Music Art of Listening Conference in Toronto, Ontario, Canada, 2005.

[11] Andreas Kornstädt. Score-to-humdrum: A graphical environment for musicological analysis. *Computing in Musicology*, 10:105–22, 1996.

[12] Andreas Kornstädt. The jring system for computer-assisted musicological analysis. In *Proceedings of the International Symposium on Music Information Retrieval*, 2001.

[13] GNU Public License. Homepage, 2006. `http://www.gnu.org/copyleft/gpl.html`.

[14] Lilypond. Homepage, 2006. `http://www.cs.wisc.edu/condor/`.

[15] Comprehensive Perl Archive Network. Cpan, 2006. `http://www.cpan.org`.

[16] PostGreSQL. Homepage, 2006. `http://www.postgresql.org/`.

[17] L. Smith, R. J. McNab, and I. H. Witten. Sequence-based melodic comparison: A dynamic-programming approach. *Computing in Musicology*, (11):101–18, 1998.

[18] Jan Stevens. Meme hunting with the humdrum toolkit: Principles, problems, and prospects. *Computer Music Journal*, 28(4):68–84, 2004.

[19] M. Taylor. Humdrum graphical user interface. Master's thesis, Queen's University, Belfast, 1996.