TuneGraph, an online visual tool for exploring melodic similarity

Chris Walshaw

Department of Computing & Information Systems University of Greenwich

London SE10 9LS, UK c.walshaw@gre.ac.uk

Abstract

This paper presents TuneGraph, an online visual tool for exploring melodic similarity. The underlying data comes from a large index of online music, all transcribed in abc notation, and TuneGraph uses a melodic similarity metric to derive a proximity graph representing similarities within the index. A rich but dense graph is built and then sparsified by removing weak, non-essential edges. From this a local graph is extracted for each vertex, aimed at indicating close variants of, and similar melodies to, the underlying tune represented by the vertex. Finally an interactive user interface displays each local graph on that tune's webpage, allowing the user to explore melodically similar tunes.

Keywords

cultural informatics; music similarity; force- directed placement; search visualisation

Introduction

-Background-

Abc notation is a text-based music notation system popular for transcribing, publishing and sharing folk music, particularly online. Similar systems have been around for a long time but abc notation was formalised (and named) by the author in 1993 (Walshaw, 1993). Since its inception he has maintained a website, now at abcnotation.com, with links to resources such as tutorials, software and tune collections.

1) Tune search engine

In 2009 the functionality of the site was significantly enhanced with an online tune search engine, the basis of which is a robot which regularly crawls known sites for abc files. The downloaded abc code is cleaned and indexed and then stored in a database which backs the search engine front end. Users of the tune search are able to view, listen to and download the staff notation, MIDI representation and abc code for each tune, and the site currently attracts around half a million visitors a year.

2) Breadth

The aim of the tune search is to index all abc notated transcriptions from across the web. However there are a number of reasons why it is unable to do this completely:

- Unknown sources: the robot indexer is seeded from around 350 known URLs (some of which are no longer active), but it does not search the entire web.
- HTML based transcriptions: in the main, the indexer searches for downloadable abc file types (.abc, or sometimes .txt). However, there are a number of sites where the abc code is embedded directly into a webpage. Mostly these tend to be small collections (especially if the abc code has to be manually inserted into the HTML code) and are ignored by the robot (although there are 3 larger collections which are included by parsing the HTML and looking for identifiable start and end tags).
- JavaScript links: for a small number of sites the file download is enacted via JavaScript, making

the link to the .abc file difficult to harvest.

Starting with an initial database of 36,000 tunes in 2009 the search engine has expanded to cover around 450,000 abc transcriptions at the time of writing (November 2014). Most of these are folk tunes and songs from Western Europe and North America, although two massive multiplayer online role- playing games, Lord of the Rings Online and Starbound, have adopted abc for their in-game music system resulting in a number of dedicated websites with mixed collections of rock, pop, jazz and, sometimes, folk melodies which contribute ~37,000 transcriptions to the search engine.

3) Duplicates & variants

Although each tune comes from a distinct URL, there are many duplicates and closely related tune variants contained within the database.

From a search engine point of view, there is little point in presenting users with dozens of identical results and so an important part of the pre-indexing clean-up involves identifying and, where appropriate, merging exact duplicates (such as those copied from one website to another – see section II) within the index.

On the other hand tune variants are an important part of folk music's aural tradition which can occur for a number of reasons (see section III) and distinct, but closely related versions of the same tune can be of interest to researchers and musicians alike. However they are not always easy to identify by eye from a large number of search results.

-Aims-

This paper discusses work which aims to address the question of how to present closely related search results to the users of a search engine. It is based on a graphical user interface developed as part of the abc notation tune search but the ideas are generic and should, in principle, be applicable to other datasets where the difference between any pair of items in the dataset can be expressed numerically (i.e. with a similarity measure).

The remainder of this paper is organised as follows:

- Section II discusses duplication and indicates how duplicates are identified and merged in the search results.
- The bulk of the work is presented in Section III which describes the development and implementation of TuneGraph, to facilitate the exploration of tune variants by users of the search engine.
- Finally Section IV presents some conclusions and future work.

Eliminating duplication

Duplication occurs widely within the abc corpus for a number of observable reasons:

- Compilations: particularly in the past, certain enthusiasts have published compilations of all the abc tunes they could find, gathered from across the web.
- Selections: some sites, usually those containing repertoires (perhaps that of a band or an open session), publish a selection of tunes gathered from other sites.
- Ease-of-access: a number of sites publish collections both as one-tune-per-file together with a single file containing all of the tunes in the collection.

As indicated above there is little point in presenting users of the search engine with duplicate results and so the pre- indexing clean-up involves identifying and merging duplicates within the index. However, it is not necessarily clear which level of duplication to remove.

-Duplicate classification-

To discuss this topic further it is helpful to consider the structure of an abc tune transcription (see 0).

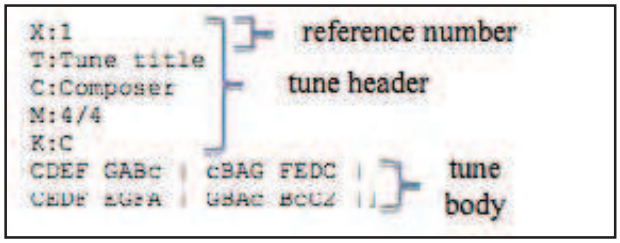

Fig. 1. An example abc transcription.

Each tune consists of a tune header (including a reference number) and the tune body.

The header contains descriptive meta-data mostly, though not exclusively, with no musical information. Typically this includes the title and composer (where known), and amongst other data may also include information about where the tune was sourced (book, recording, etc.), who transcribed it, historical notes and anecdotes and instrumentation details (particularly for multi-voice music).

The tune body contains the music, and may also contain song lyrics.

With this structure in mind, duplication can be classified into 4 increasingly broad categories:

- Electronic: the duplicates are electronically

identical (the exact same string of characters) – i.e. the tune headers and bodies are identical (although in practice this is relaxed somewhat by ignoring the reference number and any whitespace).

- Musical: the duplicates are musically identical (including song lyrics) although they may contain different meta-data in the tune header – i.e. the tune bodies are identical.
- Melodic: neglecting any song lyrics, grace notes, decorations and chord symbols, the first voice of each duplicate is identical – i.e. the primary melodies are identical.
- Incipit: when transposed to the same key, the duplicates are melodically identical over the first few bars of the tune.

In fact, and as might be expected, analysis reveals that there are no substantive differences between the musical and melodic duplicate categories and numerically there is only a 4% increase in duplication in the latter as compared with the former (Walshaw, 2014).

The other categories are substantially different, however, with 42.7% electronic duplication, 58.1% melodic duplication and 70.4% incipit duplication. (Here, the percentage duplication refers to the percentage of the corpus which can be excluded, leaving one representative example of a duplicated tune, without reducing its diversity.)

Whilst this indicates a very substantial amount of duplication within the corpus, when melodic duplicates were excluded (in a previous study, Walshaw, 2014) it gave a headline figure of 167,632 distinct melodies (out of the 400,160 under consideration), even when all of the meta-data, decorations and lyrics are stripped away.

The remainder of this paper considers only electronic duplicates and discusses ways to allow users to explore musical, melodic and incipit duplicates as tune variants.

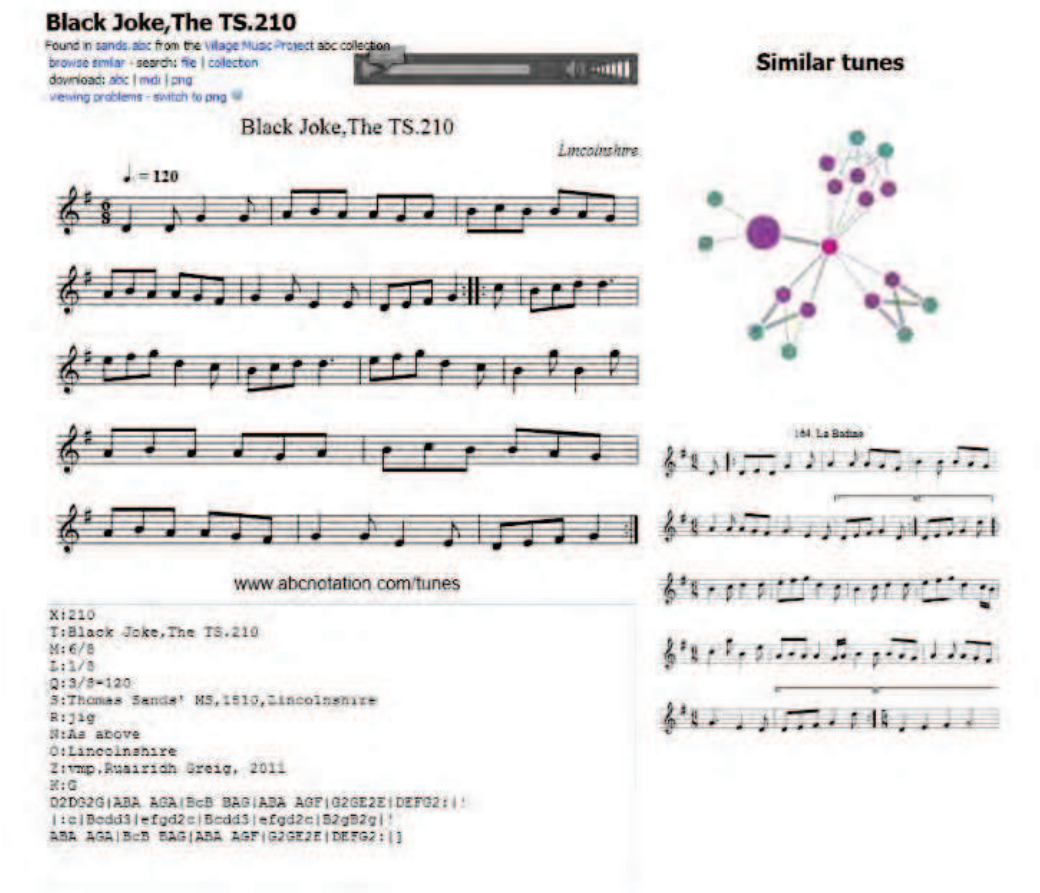

Fig 2. An example of a tune page, showing the tune in standard notation (top left), the MIDI player (top centre), the abc notation (bottom left) and the TuneGraph of close variants (top right). One of the close variants has been selected by the user (the vertex is enlarged) and is displayed below the TuneGraph viewer (bottom right).

At the time of writing of the 449,845 transcriptions in the database, 240,902 are electronic duplicates. Of the remaining 208,943 tunes, 40,179 are identified as potentially copyright. Since the abcnotation website does not display copyright tunes (unless the copyright holder has given their explicit permission) these are also excluded from the TuneGraph results, leaving a total of 168,764 under consideration.

B. Informationarchitecture

This section discusses how the data is organized and, in particular, how the search engine distinguishes between duplicates, which are not presented in standard search results, and tune variants, which are.

information architecture is as follows.

1) Duplicates

One possibility would be to completely remove duplicates from the database. However, this would mean that if, for example, users filter their search to look at a particular source website they will miss all the duplicates offered by that site. It also gives a misleading impression of the contribution from each site.

Instead tunes are categorised into primary and secondary sources. Thus a cluster of n duplicates would contain 1 primary representative and n – 1 secondary.

Using this categorisation, standard search results only include primary tunes, but when a user clicks through from the search results to a tune page (each tune in the database has its own page), a list of secondary sources for that tune is also included (since some tunes can have many secondary sources this list is restricted to a maximum of 10 randomised entries).

If, however, the user filters their results to a

particular source website (for example, the search query term "site:www.example.com/folder" would restrict the search results to tunes from http://www.example.com/folder/ and any subfolders) then both primary and secondary tunes are listed in the search results.

The scheme (not discussed here) for categorising sources into primary and secondary is based on their originating URL and the author's somewhat subjective view of where a tune was first published on the web.

2) Tune variants

Tune variants are an important part of folk music's aural tradition which can occur for a number of reasons, including:

- Transmission: folk music is an aural tradition – a tune may be misheard or misremembered when it is passed on
- Improvisation: many traditions have an improvisatory aspect
- Innovation: musicians may devise their own versions of a tune

As a result of these and other reasons the same tune may have developed differently in different geographical locations over a period of time.

These may be of interest to researchers and musicians alike. However they are not always easy to identify by eye from a large number of search results.

It is not obvious how close variants should be presented and one possibility might be to show, somehow, a representation of variants on search results pages.

However, it was felt that this would clutter the results too much and so instead each tune page shows a representative group of variants in a small(ish) interactive graphic, the TuneGraph viewer (see section III.B) on the right hand side of the page (see Fig. 2).

The following section discusses how the variants are derived and the user interface for the graphic.

Tunegraph – exploring variants

The aim of TuneGraph is to facilitate user exploration of tunes variants.

Given a corpus of melodies, the idea behind it is to calculate the difference between each pair of melodies numerically with a difference metric or similarity measure (e.g. Kelly, 2012; Stober, 2011; Typke, Wiering, & Veltkamp, 2005).

Next a proximity graph is formed by representing every tune with a vertex and including (weighted) edges for every pair of vertices which are "similar". Finally, the resulting graph can be visualised using standard graph layout techniques such as force-directed placement, (e.g. Walshaw, 2003), either applied

to the entire graph or just, as here, to a vertex and its neighbours (i.e. a tune and similar melodies).

The concept is not dissimilar to a number of other software systems which give a visual display of relationships between tunes, often based on a graph (e.g. Langer, 2010; Orio & Roda, 2009; Stober, 2011).

The TuneGraph software consists of two parts – TuneGraph Builder (Section III.A), which analyses the corpus and constructs the required graphs, and TuneGraph Viewer (Section III.B), which provides the online and interactive visualisation.

A. TuneGraphBuilder

1) The similarity measure

In the current implementation, each melody is represented by quantising the first 3 bars (the incipit) into 1/64th notes and then constructing a pitch vector (or pitch contour) where each vector element stores the interval, in semitones, between the corresponding note and the first note of the melody (neglecting any anacrusis). Since everything is calculated as an interval it is invariant under transposition.

The similarity measure or difference metric then calculates the difference between two pitch vectors either using the 1-norm (i.e. the sum of the absolute values of the differences between each pair of vector elements) or the 2-norm (i.e. the square root of the sum of squared differences between each pair of vector elements). The 1-norm difference metric has long been available as part of the abc2mtex indexing facilities (Walshaw, 1994), but experimentation suggests that the 2-norm gives marginally better results (see below, section (7)). If the pitch vectors have different lengths then the sum can be calculated over the length of the shorter vector (although see below – section (3)).

Similarity measures of this kind are well explored in the field of music information retrieval, (e.g. Kelly, 2012; Typke et al., 2005), and there may be other, more advanced similarity measures that would work even better. However, in principle any suitable metric can be used to build the proximity graph, provided that it expresses the difference between pairs of melodies with a single numerical value. Indeed, even combinations of similarity measures could be used by forming a weighted linear combination of their values.

2) Building the proximity graph

The proximity graph is formed by representing every tune with a vertex and including (weighted) edges for every pair of vertices which are "similar" (i.e. every pair where the numerical difference is below some threshold value). However the question arises: what is a suitable threshold and how should it be chosen?

Perhaps the simplest choice, and one which is well-known for geometric proximity graphs, is to find the smallest threshold value which results in connected graph, i.e. a graph in which a path exists between every pair of vertices. Although computationally expensive, this can be done relatively straightforwardly starting with an initial guess at

a suitable threshold and then either doubling or halving it until a pair of bounding values are found, one of which is too small (and does not result in a connected graph) and one of which is large enough (and does give a connected graph). Finally the minimal connecting threshold (minimal so as to exclude unnecessary edges) can be found with a bisection algorithm, bisecting the interval between upper and lower bounds each iteration.

This was the first approach tried but it resulted in graphs with an enormous number of edges: the test code ran out of memory as the number of edges approached 200,000,000 and the threshold under test had not, at that point, yielded a connected graph.

Further investigation revealed the basic problem: the graph is potentially very dense in some regions, with many similar melodies clustered together, whereas elsewhere there are outlying melodies which are not similar to any others. This means that in order to connect the outliers, and hence the entire graph, the threshold has to be so large that in the denser regions huge cliques are generated.

3) Segmentation by meter

In order to reduce the density of the graph, one successful approach tested was to segment the graph by meter – i.e. so that tunes with different meters are never connected. In fact a simple way to implement this is to avoid connecting pitch vectors with different lengths. This has the added benefit that some meters can be connected (i.e. those with the same bar length such as 2/2 and 4/4) meaning that the strategy is blind to certain variations in transcription preferences (although not universally as it will fail to connect related melodies, such as Irish single jigs, which are variously transcribed in 6/8 and 12/8, and French 3-time bourrées, which can be either 3/4 or 3/8).

Each pitch vector length results in a subset of graph vertices: in all, for the 168,764 tunes under consideration, there were 137 subsets, ranging in size from 65,568 vertices (for 2/2 and 4/4 tunes), down to 60 subsets containing just one vertex. However, 99.46% of vertices are in a subset of size 100 or more and 99.85% are in a subset of size 10 or more.

The small subsets generally result from unusual vector lengths, usually because of errors in the transcriptions (i.e. extra notes or incorrect note lengths) and there was often no close relation between the melodies, meaning that a very high threshold would have to be used to connect that subset. To avoid connecting very different transcriptions, for each segment the edge threshold was, somewhat arbitrarily, limited to the length of the pitch vector for that segment. In most cases, this upper limit was never needed, but for very small subsets it sometimes meant that no edges were generated at all.

4) Target median degree

Even with segmentation by meter in place the method can still generate graphs with huge numbers of edges. However, there is no particular reason that the graph needs to be connected so the idea of trying to build a connected graph (or connected sub-graphs, one for each pitch vector length) was abandoned as impractical. Nevertheless, it is attractive as essentially parameter-free and it does work for small collections of relatively closely related tunes (for example, English morris tunes, where there are many similar variants of the same melody).

For the purposes of representing the entire corpus as a (disconnected) proximity graph, this still leaves the choice of a suitable edge threshold open. Rather than picking a value out of the air, instead a target average degree for the resulting graph is determined by experimentation. With this average degree as a parameter the same bounding and bisection method as above (section (2)) can be used to find the smallest threshold that yields this average degree.

An important observation is that the small number of vertices which have very many similar neighbours generate a relatively large number of edges in the graph. For example a cluster of, say, 100 very similar melodies will form a (near) clique with up to 4,950 edges. This significantly skews the average if it is expressed as the mean degree. However, the median degree ignores these outlying values and gave much more useful results empirically and so the TuneGraph Builder uses the target median degree, D. Considerable experimentation has been carried out with a number of target median degree values with the aim of finding one which yields a large number of local graphs that are small enough to be useful in search but which are sufficiently rich enough to express similarities visually (see below, section (7)).

5) Sparsification

Experimentation also revealed that, on its own, the use of the target median degree to decide which edges to include is far too crude.

An alternative approach which proved much more successful is to build a rich, and hence very dense graph initially and then sparsify it by removing the weakest "non-essential" edges. The advantage of this approach is that in effect it provides a variable threshold for including edges: in regions where the graph is dense, many edges are removed. However, in areas where the graph is already sparse, edges are retained even if they are weak, if they are deemed to be essential.

The algorithm designed to achieve this sparsification turned out to be very simple but also extremely effective. All the edges are added to a list and sorted primarily by combined degree (if edge e is incident on vertices u and v then the combined degree of e is the degree of u plus the degree of v), largest to smallest, and then by weight, smallest to largest. This roughly prioritises the densest regions and within them, the weakest edges.

To sparsify the graph the list is traversed and each edge encountered is removed from the graph if both of its incident vertices have degree greater than a pre-specified

minimum sparsification degree parameter, S.

For example, if S = 3 then an edge is removed if both of its incident vertices have degree of 4 or more. Since the degrees of vertices are updated during the sparsification process this means that once a vertex is reduced to a degree of S then no more of its edges can be removed.

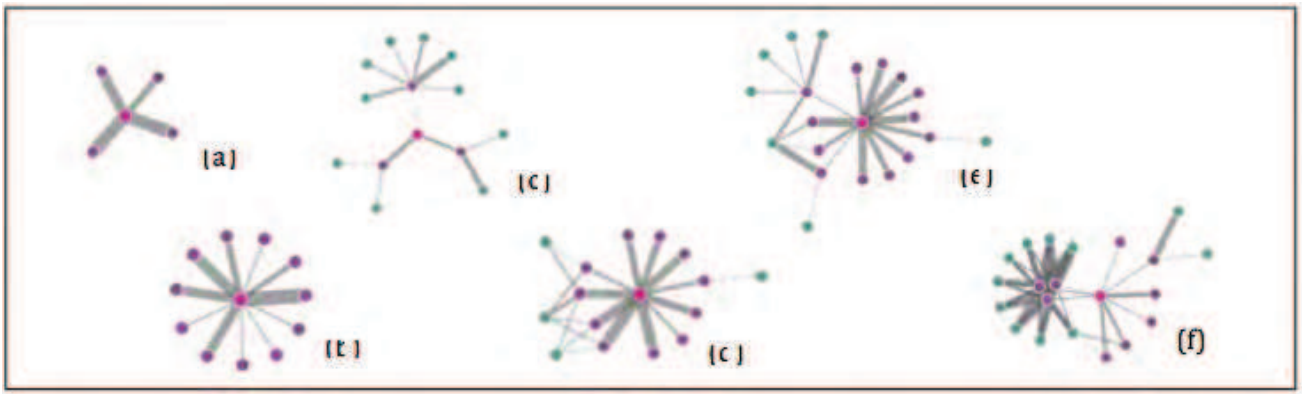

Fig. 3. Some sample local graphs.

6) Extracting local graphs

Having built and subsequently sparsified the graph the TuneGraph Builder code extracts a local graph for each non- isolated vertex (the local graph is what will ultimately be displayed alongside the tune represented by that vertex).

One way to do this is simply to extract the vertex, plus all its neighbours plus any edges between them. However, this can lead to clique-like local graphs where edges are hard to discern.

Instead, the local graph is built in layers: the seed (layer 0) is the original vertex for which the local graph is being built, layer 1 is any vertices neighbouring layer 0 and layer 2 is any vertices (not already included) neighbouring layer 1, etc. In order to maximise the clarity of the local graph, it only includes edges between layers and excludes edges between vertices in the same layer.

Fig. 3 shows some examples: Here (a) and (b) come from local clique-like graphs with no immediate neighbours (recall that edges between vertices in the same layer are not included in the local graph so not all edges of the clique are shown). The tree shown in (c) indicates a number of tunes which are related but probably not immediate relations of each other. The graphs in (d) and (e) are similar to (b) only with some outlying tunes related to those in the clique. Finally the graph in (f) shows a tune, with many variants, on the edge of a tightly coupled clique.

If the local graphs are just built from layers 0 and 1, each will be star-like, as in Fig. 3(a) and Fig. 3(b), yielding limited immediate visual information to the user (other than the number of neighbours and the strength of the relationships). Instead the builder code uses layers 0, 1 and 2, e.g. Fig. 3(c) to Fig. 3(f), to show some of the richness of certain neighbourhoods. Here colours indicate the layers, with layer 0 shown in crimson, layer 2 in light blue, and layer 1 interpolated between the two of them.

Finally, the graph edges are all weighted in inverse proportion to the difference between the two tunes that they connect. Since graph edge weights are indicated in the online tool by their thickness this conveys helpful information to the user by showing the more closely related tunes with thicker lines between them (and also affects how the graph is laid out by force directed placement).

7) Experimentation and parameter selection

It is difficult to say exactly what features are desirable in the visualisations provided for users, but experience with suggests that the local graphs should be small enough not to overwhelm the user, but rich enough to convey some useful information. In particular the aim was to limit the maximum local graph size but maximise the average size.

Experimentation was carried out with a number of different parameter settings but it is not at all easy to decide which are the best parameter settings to use and therefore a simple scoring system was employed.

Based on (subjective) analysis of many example graphs, the following principles were established:

- graphs with 20 or fewer vertices are the most easy to assimilate and use; above 40 or so vertices they start to become over-crowded and as they approach 100 vertices they are virtually unusable (at least in the space allowed for them on the web page);
- star graphs, e.g. Fig. 3(a) and Fig. 3(b), are less interesting than layered graphs, e.g. Fig. 3(c) to Fig. 3(f).

With these in mind the following scoring system was implemented:

1 point for each star graph with up to 40 vertices

0 points for each star graph with 41-60 vertices -1 point for each star graph with 61+ vertices

2 points for each layered graph with up to 20 vertices 1 point for each layered graph with 21-40 vertices

0 points for each layered graph with 41-60 vertices

-1 point for each layered graph with 61-80 vertices -2 points for each layered graph with 81+ vertices

The two most crucial parameters were found to be D, the target median degree (section (4), used to find a suitable proximity threshold for adding edges) and S, the sparsification minimum degree (section (5), used as a lower limit when removing edges). These are interdependent: D determines the richness of the overall graph whilst S determines how many of the weaker edges are removed.

Of the two S is the cruder control. If S = 1 then all of the local graphs end up as star graphs (and so the overall score is considerably reduced). However, as S is increased (and hence sparsification decreased), the average size of the local graphs increases accordingly and the overall score is negatively impacted by increasing numbers of large local graphs regardless of which value for D is chosen.

Thus, and perhaps surprisingly, the best value for S is 2. 0shows some example results giving the overall score for selected values of S and D. For S = 2, where the best scores were obtained, the very best choice of D was 28, but in fact there are several scores close by and all values of D between 24 and 32 (not all shown in the table) yield scores over 229,000. Even when D = 45, the score is still over 227,000.

	S = 1	*S* = 2	*S* = 3	*S* = 4
D = 10	133,432	210,430	207,573	198,140
D = 15	138,906	221,248	216,589	**200,535**
D = 20	141,915	227,229	**218,551**	196,382
D = 25	143,590	229,177	217,148	191,005
D = 26	143,594	229,194	217,133	190,964
D = 27	143,629	229,260	216,919	190,692
D = 28	144,310	**229,903**	215,939	187,701
D = 29	144,618	229,843	215,203	185,915
D = 30	145,039	229,197	213,270	181,658
D = 35	145,342	228,363	210,222	176,267
D = 40	145,955	228,290	207,152	170,785
D = 45	**146,636**	227,531	201,983	162,379

As S increases the scores drop off rapidly, particularly as D increases. 0illustrates the reasons why by taking a closer look at the graphs produced for D = 28 and different values of S. In each case, 145,594 local graphs are produced and the table then breaks them down into categories by size and by type, star and layered.

Perhaps the best way to view this is by looking at S = 4 (right hand column) first of all. The majority of graphs here are layered but nearly 15,000 of them are above 40 in size, contributing nothing or even negative points to the score. When S is decreased to 3 around 9,000 of these are reduced in size and end up either as smaller star graphs (~5,000) or smaller layered graphs (~4,000).

When S is reduced to 2 a further 6,000 large layered graphs are removed, probably becoming smaller star graphs.

Even more importantly the decrease of S transfers a large number of layered graphs from the 21 – 40 category into the 1 – 20 category, doubling the score for them.

	sizes	*S* = 1	*S* = 2	*S* = 3	*S* = 4
star	1 – 20	137,038	**37,215**	24,936	21,142
	21 – 40	7,408	**2,453**	2,930	1,842
	41 – 60	1,012	**66**	357	400
	61 – 80	136	**2**	18	55
	81+	0	**0**	2	3
layered	1 – 20	0	**86,660**	77,526	60,622
	21 – 40	0	**17,063**	33,825	46,396
	41 – 60	0	**1,989**	5,323	12,559
	61 – 80	0	**146**	570	2,285
	81+	0	**0**	107	290
score		144,310	**229,903**	215,939	187,701

Finally, as mentioned above, there are no layered graphs produced when S = 1 so that even though the number of graphs in the 1 – 20 category is the highest of all 4 S values, the total is lower as the scoring system favours layered graphs. A large number of other tests were carried out, not presented here. However from these the following final parameter settings were chosen:

- Difference norm: ||.||2 – see section (1)
- Segmentation by meter: true – see section (3)
- Edge threshold limit: pitch vector length – see section (3)
- Target median degree: 28 – see section (4)
- Minimum sparsification degree: 2 – see section (5)

In all experiments, regardless of parameter settings, there were a residue of isolated vertices, usually because there are no closely related melodies in the corpus or, less commonly, because there are no other transcriptions with the same pitch length. Eliminating these isolated vertices gives a final graph (for the chosen parameter settings) of

145,594 vertices.

Prior to sparsification the graph had 7,139,396 edges, maximum vertex degree of 2,060 and an average degree of 98.07; afterwards these figures were reduced to 204,639 edges, with a maximum degree of 77 and an average of 2.81, indicating the success of the sparsification algorithm.

The sparsified graph is less connected than the pre-sparsified one and the former contains 10,536 connected subsets (many with as few as 2 vertices) as compared with 5,616 connected subsets in the latter. However, a connected graph was not an aim of the process (particularly since the graph is already segmented by meter).

From this global graph 145,594 local graphs were produced with an average size of 12.3 vertices. The maximum size was 80 vertices and 154 edges.

Finally it should be emphasized that these figures are just a snapshot taken at the time of writing (November 2014) and indeed differ significantly from the prototype version layered star presented previously (Walshaw, 2014) which did not use sparsification. Furthermore, the robot which gathers data for the tune search is run every month and each time the abc files available change, meaning that so too will the underlying graph and the number of local graphs produced. The choice of parameters is chosen with the current data in mind but it is likely that the highest scoring choice of target median degree, D, may change over time according to the underlying data.

It is also likely that the scoring system will be modified as users' impressions of the local graphs are assessed.

B. TuneGraphViewer

The TuneGraph Viewer has been deployed on the abcnotation.com website since 1st September 2014 and provides the an interactive user interface for viewing each local tune graph (on a webpage alongside the tune it corresponds to).

The local graph is visualised as a dynamic layout using D3.js (Bostock, 2012), a JavaScript library for manipulating documents based on data, and employing the inbuilt force- directed placement features.

It provides the following user interface:

- The graph vertices find their own natural positiondynamically via force directed placement and vertices can be dragged to rearrange the layout (other vertices then relocate accordingly).
- Vertex colour indicates the relationship to the root vertex (with layer 0 shown in crimson, layer 2 in light blue, and layer 1 interpolated between the two of them).
- Edge thickness indicates visually how closely related two vertices are (i.e. how similar their corresponding tunes are).
- Moving the mouse over a vertex reveals its name and displays the associated melody.
- Double clicking on a vertex (other than the root vertex) takes the user to the corresponding page (with its own tune graph).

Fig. 2 shows an example webpage corresponding to the tune Black Jack (a well-known English tune). The tune is displayed on the left with the abc notation underneath and the local tune graph is shown on the right. When the user moves their mouse over one of the graph vertices, the interface enlarges the vertex and notation for the tune associated with that vertex appears below.

Conclusion

This paper has presented TuneGraph, an online visual tool for exploring melodic similarity.

It is based upon a large index of online music and uses a melodic similarity measure to derive a proximity graph representing similarities within the index.

A rich but dense graph is built and then sparsified by removing weak non-essential edges. From this a local graph is extracted for each vertex, indicating close variants and similar melodies of the underlying tune represented by the vertex. Finally an interactive user interface display each local graph is on that tune's webpage, allowing the user to explore melodically similar tunes.

A. Futurework

The main focus for future work is to enhance the capabilities of TuneGraph. In particular it is intended to explore some of the wide range of similarity measures that are available as a means to build the proximity graph. As was indicated in section III.A there may be other, more advanced similarity measures, or combinations of similarity measures, that would work better than the 2-norm of the difference between pitch vectors.

Furthermore, at this point the similarity measure used to assess the proximity of variants is based on the incipit only (first 3 bars, neglecting any anacrusis) and at some point in the future it is intended to use a more discerning metric based on much larger portions of the tune (as not all closely related incipits are as a result of closely related tunes).

References

1. Bostock, M. 2012. Data-Driven Documents (d3.js), a visualization framework for internet browsers running JavaScript. Retrieved May 15, 2014, from http://d3js.org/
2. Kelly, M. B. 2012. Evaluation of Melody Similarity Measures. Queen's University, Kingston, Ontario.
3. Langer, T. 2010. Music Information Retrieval & Visualization. In Trends in Information Visualization, pp. 15–22.
4. Orio, N., & Roda, A. 2009. A Measure of Melodic Similarity

based on a Graph Representation of the Music Structure. Proc. ISMIR, pp. 543–548.

5. Stober, S. 2011. Adaptive Distance Measures for Exploration and Structuring of Music Collections, Section 2, 1–10.
6. Typke, R., Wiering, F., & Veltkamp, R. C. 2005. A survey of music information retrieval systems. In Proc. ISMIR, pp. 153–160.
7. Walshaw, C. 1993. ABC2MTEX: An easy way of transcribing folk and traditional music, Version 1.0. University of Greenwich, London.
8. Walshaw, C. 1994. The ABC Indexing Guide Version 1.2. University of Greenwich, London.
9. Walshaw, C. 2003. A Multilevel Algorithm for Force-Directed Graph-Drawing. Journal of Graph Algorithms and Applications, 73, pp. 253–285.
10. Walshaw, C. 2014. A Statisical Analysis of the Abc Music Notation Corpus. In A. HolzapfelEd., Folk Music Analysis, Istanbul, pp. 2–9. Istanbul: Bogaziçi University.