A HIERARCHIC DIFF ALGORITHM FOR COLLABORATIVE MUSIC DOCUMENT EDITING

Christopher Antila
nCoda
christopher@antila.ca

Jeffrey Treviño
California State University, Monterey Bay
jtrevino@csumb.edu

Gabriel Weaver
University of Illinois, Urbana-Champaign
gweaver@illinois.edu

ABSTRACT

We describe an application of hierarchic diff to the collaborative editing of tree-based music representations, using Zhang and Shasha's tree edit distance algorithm as implemented within the XUDiff tool. The edit distance between two trees is the minimum number of edit operations necessary to transform one tree into the other. We consider common operations on the score tree—deleting, changing, and appending tree nodes—to derive a minimal edit sequence, known as an edit script, and we compare the performance of the widely used Longest Common Subsequence algorithm against our approach. We conclude by summarizing implications for the design of collaborative music document software systems.

1. INTRODUCTION

1.1 Collaborative Document Creation Requires Diff Algorithms

In distributed, collaborative document creation, multiple editing agents may change the same original information simultaneously, in complex and overlapping ways. To allow users to resolve conflicting edits and to create a reliable and transparent edit history, robust systems for collaborative editing often depend on *Centralized Version Control Systems* (also known as *Revision Control Systems (RCS)* and *Source Code Management (SCM)* systems). These systems maintain a centralized information representation (*repository*) and a history of users' changes to it; the difference between two edited versions of the information is known as a *diff*, and this difference can be calculated and represented in various ways. Distributed, collaborative music document editing presents unique challenges for the implementation of version control systems, and especially for the implementation of a diff algorithm.

1.2 The Longest Common Subsequence Algorithm as Default Diff

Traditional document comparison algorithms, such as in the Unix `diff` program, take two sequences of characters as input and output an edit script to transform one sequence into the other. An *edit script* consists of a sequence of edit operations (usually insert, delete, and update) to transform the first sequence relative to some entity—usually characters, words, or lines. The *edit distance* is the minimum cost the edit script gives for each operation. The *Longest Common Subsequence algorithm* and its variants are the most common for computing an edit script and cost.

The Longest Common Subsequence algorithm works well in situations where the inputs are sequences of characters and one needs to compare those sequences relative to characters, words, or lines, but many modern file formats rely on hierarchical object models to encode multiple levels of meaning (e.g. XML, blocks of code). As such, different algorithms for hierarchical structures become necessary.

Consider the following two problems that result from this mismatch between character- or line-based diff tools and tree-structured input. First, comparing documents in terms of low-level entities (e.g. lines) may not result in changes that are meaningful to the domain, because lines are often an artifact of presentation: for example, one can generate 'noisy diffs' by just changing whitespace. Second, the manner in which one defines document similarity may change depending upon the task at hand. A poet may get along fine comparing texts in terms of lines, which reflect part of the structure of the text. A musician, however, may want to compare documents in terms of additional information that a line-based approach discards. Our poet, after all, may require a stanza-based representation. Other communities present similarly various demands: scholars may want to analyze and compare texts relative to other structures, such as paragraphs or sections.

1.3 Collaborative Music Information Requires a Hierarchic Diff

These problems are of specific importance to version control for collaborative music document editing, both in terms of usability as well as how one may want to define 'meaning' and 'similarity' in musical information. First, the 'noisy diff' problem—in reporting differences that are not relevant to musicians—creates a usability problem. Although programmers have become accustomed to noisy diffs and the work-arounds they require, the low adoption rate of computer-driven music analytic tools, and the general lack of comfort among music scholars and artists with these tools, suggest that a program producing diffs of meaningless changes would be poorly received by the community. Second, the meaning of textually encoded music always requires additional interpretation, and a one-dimensional

sequence of characters (the data structure for which LCS diff was designed) will not allow musicians to compare two different interpretations of the same musical information.

Instead, musicians need the ability to compare musical information in the presence of its logical organization, which must be expressed hierarchically. Therefore, musicians need the ability to compare two versions of a hierarchical structure. Musicians may also want to compare two versions of a score at different levels of abstraction, as represented by these hierarchical structures, or restrict comparison to entities with certain properties: for example, a musician may want to compare two versions of a score in terms of pitch class alone, or of higher-level features like phrase structure. Moreover, a musician may want to filter a score to compare two versions of only a single instrument's staff, or other musical abstractions. For any of this to be possible, diff algorithms must compare edits to tree-based document elements, rather than to document lines or characters.

1.4 Relevant Precedents in the Music and Computer Science Literatures

From the perspective of computer science, our proposed approach leverages previous work from other domains in both industry and in academia. Within industry, there are proprietary, hierarchy-aware difference engines that compare source code in a variety of edit operations, such as the SmartDifferencer by Semantic Designs [1]. Within academia, the tree diffing problem has been long studied by theoretical computer science [2]. Researchers such as Chawathe et al. and Cobena et al. have studied alternative algorithms, such as subtree hashing, and even the use of IDs to align subtrees before similarity computation [3, 4]. We employ the Zhang and Shasha tree difference algorithm to solve the edit distance between trees [5, 6].

While this topic has been approached rigorously in the computer science literature, few relevant precedents exist in the music literature. Almost all software systems for music document creation assume a single user, and collaborative music document creation exists largely as an experimental pursuit. Precedents fall largely into the category of systems for experimental music and interdisciplinary artistic collaboration. Within these experimental systems, version control for distributed workflows has been addressed only implicitly. For example, although Wüst and Jordá track versions in a recursively nested, tree-based structure, in which each successive edit becomes a child node of the version edited, they do not describe algorithms or interfaces for calculating edit distances on this version tree [7]. Likewise, other systems describe distributed collaboration interfaces without addressing the interaction of data representation and version comparison [8, 9].

2. APPLYING THE ZHANG AND SHASHA ALGORITHM

2.1 The Zhang and Shasha Algorithm

Our initial approach uses Zhang and Shasha's tree edit distance algorithm, as implemented within the `XUDiff` tool [10]. The edit distance between two trees is the minimum number of edit operations necessary to transform one tree into another. The edit operations we consider include deleting, changing, and appending tree nodes. As before, a sequence of edit operations between two trees is called an *edit script* and the total number of edits the *edit distance*.

There are several benefits to the Zhang and Shasha tree edit difference algorithm. First, the algorithm produces a tree edit distance that can function flexibly as a metric (assuming the cost function is also a metric). For example, collaborators can use distance metrics to explore the similarity of an entire corpus of musical scores—rather than just two scores—because the metric's notion of distance aligns with our intuition about distance in the physical world. This distance metaphor allows designers to leverage existing musical research in topological feature similarity metrics, which expands the algorithm's utility beyond the notion of hierarchic diff, into new applications such as automated recommendations based on similarity measures [11, 12, 13]. Second, the algorithm is relatively simple and lends itself to a straightforward implementation that can be maintained by an open-source community. The intent of the open-source community is to support an extensible framework for hierarchical comparison of a wide variety of document types across a number of domains. In the future, other algorithms, such as those mentioned above, may be implemented to understand more about the effect of different tree-edit distance algorithms on similarity results.

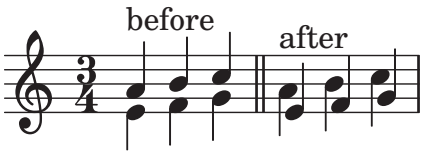

Figure 1. An edit that switches the first and second voices in a staff. Stem direction is the only visual difference, but the underlying representation changes substantially.

2.2 A Comparative Example

Consider the case of a simple edit: exchanging a staff's two voices. That is, as shown in Figure 1, the upward-facing stems of voice one become the downward-facing stems of voice two, and vice versa.

While Common Western Notation displays only a change of stem direction, a tree-based, hierarchic representation of this musical information must alter both the labeling and succession of elements. In the MEI XML representation of a music document, the voice-switch example may be encoded in the following way:

```
<staff n="1">
    <layer n="1">
        <note pname="a"/>
        <note pname="b"/>
        <note pname="c"/>
    </layer>
    <layer n="2">
        <note pname="e"/>
        <note pname="f"/>
        <note pname="g"/>
    </layer>
</staff>
```

After the voice swap, the encoding becomes:

```
<staff n="1">
    <layer n="1">
        <note pname="e"/>
        <note pname="f"/>
        <note pname="g"/>
    </layer>
    <layer n="2">
        <note pname="a"/>
        <note pname="b"/>
        <note pname="c"/>
    </layer>
</staff>
```

2.3 Diff Computation Performance Comparison

This example, although basic, motivates the need to to compare representations of music in terms of hierarchical structure, rather than lines or characters. Figure 2 illustrates an edit script that maps one version of the above MEI-encoded score to another in terms of lines (LCS algorithm). The line-based approach successfully captures the need to exchange the notes between layers; however, the algorithm adds additional noise, because `diff` compares the MEI rather than the hierarchical structure encoded by the MEI. As a result, the total edit distance is 10. If practitioners are interested in understanding change relative to the hierarchical object model of MEI, they will need to sift through the noisy changes produced by a line-based comparison. As mentioned earlier, this may be problematic for widespread adoption within the music community.

In contrast, Figure 3 illustrates an edit script that maps one version of the above MEI-encoded score to another in terms of MEI's hierarchical object model (Zhang and Shasha algorithm). As with the LCS algorithm, the tree-based approach successfully captures the need to exchange the notes between layers; however, unlike the LCS algorithm, the edit distance between individual subtrees has also been summarized. This can be helpful for interpretation, as subtrees closer to the root represent higher-level constructs within MEI, and practitioners can interpret the comparison of the music at multiple levels of abstraction, ranging from low-level notes (six `notes`, each with an edit cost of 1) to higher-level layers (two `layers`, each with an edit cost of 3) and staves (one `staff`, with an edit cost of 6). Most notably, the tree-based element-by-element comparison reduces the edit distance to almost half of that of the LCS algorithm: the edit distance has been reduced to 6, from LCS's 10, most of which was noise from deleting lines with 'layer' tags (an artifact of line-based comparison).

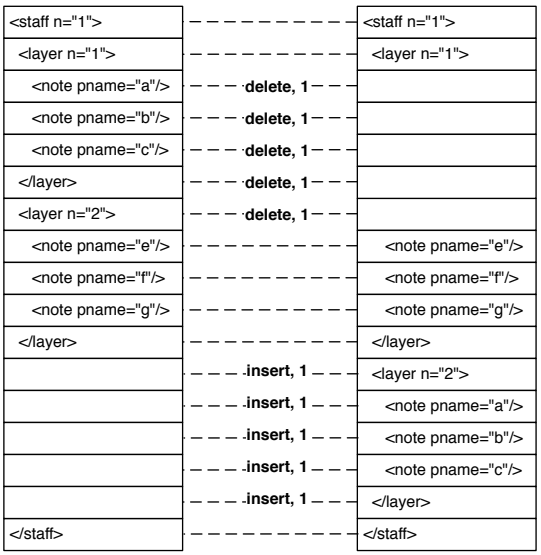

Figure 2. The figure above illustrates the output of `diff` applied to MEI. A total edit distance of 10 results from updating the notes in layer 1 and layer 2 (cost of 6), as well as updating `layer` tags (cost of 4). Nearly half of the edit distance is 'noise' from deleting lines with `layer` tags, an artifact of comparing versions in terms of lines instead of MEI elements.

3. CONCLUSIONS

The recently emerged potentials of online, collaborative music applications illustrate several ways that a robust, hierarchic diff algorithm for music can enable newly collaborative musicology, composition, and music education through document utilities [14, 15, 16, 17]. Yet the commercial presentation of widely used digital engraving tools still conflates the act of sharing with the act of collaboration, although these remain distinct from each other. As a recent advertisement for the Sibelius engraving program exhorts, 'Collaborate more easily with others and distribute your compositions for the world to hear. Share scores through email, upload and publish them as sheet music on ScoreExchange.com, and even share your composition as a video or audio file on YouTube, Facebook, and SoundCloud' [18]. While file exchange between music authors remains crucial for musical creativity and collaboration beyond notation, it is time for engraving software to embrace the potentials of genuinely collaborative music document editing interfaces. But distributed music document collaboration requires robust, intuitive version control algorithms and interfaces, and designers must reassess the task of music representation in light of the need for hierarchic diff. The superior performance of the Zhang

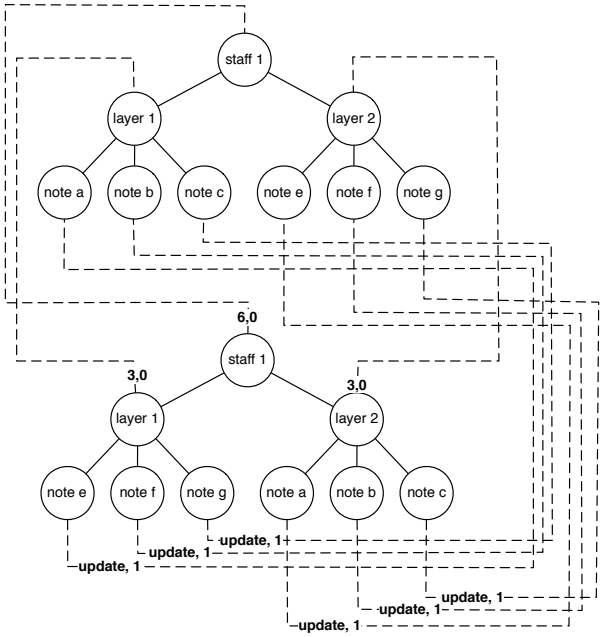

Figure 3. The figure above illustrates the output of `xudiff` applied to MEI. A total edit distance of 6 results from updating the notes in layer 1 and layer 2. Total costs are aggregated across the hierarchical structure of the MEI text.

and Shasha algorithm shown here suggests that purely tree-based representations, such as MEI, should be adopted for collaborative music software systems.

Acknowledgments

Research conducted for the *nCoda* project has been supported by Colorado College's SEGway faculty support grant.

4. REFERENCES

[1] S. Designs, "Semantic Designs: Smart Differencer Tool." [Online]. Available: http://www.semdesigns.com/Products/SmartDifferencer/

[2] P. Bille, "A Survey on Tree Edit Distance and Related Problems," in *Theoretical Computer Science*, vol. 337, June 2005, p. unknown.

[3] S. Chawathe *et al.*, "Change Detection in Hierarchically Structured Information," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '96)*, June 1996, pp. 493–504.

[4] G. Cobéna *et al.*, "Detecting Changes in XML Documents," in *Proceedings of the 18th International Conference on Data Engineering*. IEEE, February and March 2002, pp. 41–52.

[5] K. Zhang and D. Shasha, "Simple Fast Algorithms for the Editing Distance between Trees and Related Problems," *Siam Journal of Computing*, vol. 18, pp. 1245–1262, December 1989.

[6] K. Zhang, "The Editing Distance between Trees: Algorithms and Applications," Ph.D. dissertation, New York University (NYU), 1989.

[7] O. Wüst and S. Jordà, "Architectural Overview of a System for Collaborative Music Composition over the Web," in *Proceedings of the 2001 International Computer Music Conference*. Citeseer, 2001, pp. 298–301.

[8] S. Balachandran and L. Wyse, "Computer-mediated Visual Communication in Live Musical Performance: What's the Score?" in *Arts and Technology*, A. L. Brooks, Ed. Springer, 2012, vol. 101, pp. 54–62. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33329-3_7

[9] D. Hepting and D. Gerhard, "Collaborative Computer-aided Parameter Exploration for Music and Animation," *Computer Music Modeling and Retrieval*, pp. 158–172, 2005. [Online]. Available: http://www.springerlink.com/index/XJXLV487NLUU9W9V.pdf

[10] G. Weaver, "Security-Policy Analysis with eXtended Unix Tools," Ph.D. dissertation, Dartmouth College, 2013.

[11] J. P. Bello, "Measuring Structural Similarity in Music," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 7, pp. 2013–2025, 2011.

[12] A. Berenzweig, B. Logan, D. P. Ellis, and B. Whitman, "A Large-scale Evaluation of Acoustic and Subjective Music-similarity Measures," *Computer Music Journal*, vol. 28, no. 2, pp. 63–76, 2004.

[13] Y. Panagakis and C. Kotropoulos, "Music Genre Classification via Topology Preserving Non-negative Tensor Factorization and Sparse Representations," in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2010, pp. 249–252.

[14] D. Martin *et al.*, "LeadsheetJS: A Javascript Library for Online Lead Sheet Editing," in *Proceedings of The First International Conference on Technologies for Music Notation and Representation*, 2015.

[15] P. McCulloch, "THEMA: A Music Notation Software Package with Integrated and Automatic Data Collection," in *Proceedings of The First International Conference on Technologies for Music Notation and Representation*, 2015.

[16] Flat, "flat.io." [Online]. Available: flat.io

[17] T. Bača, J. Oberholtzer, J. Treviño, and V. Adán, "Abjad: An Open-source Software System for Formalized Score Control," in *Proceedings of The First International Conference on Technologies for Music Notation and Representation*, 2015.

[18] Avid, "Sibelius: Features." [Online]. Available: http://www.avid.com/sibelius/features