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 initial approach to hierarchic diff for collaborative music document editing via 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 another. 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.

1. INTRODUCTION

1.1 The Least Common Subsequence Algorithm as Default Diff

Traditional document comparison algorithms, such as the Unix diff, takes two sequences of characters as input and outputs an edit script to transform one sequence into another. An edit script consists of a sequence of edit operations (usually insert, delete, and update) to modify transform the first sequence relative to some type of entity such as characters, words, or lines. The edit distance is the minimum cost edit script given costs for each operation. Typical algorithms to compute the script and the cost are variants of the Least Common Subsequence algorithm.

This approach to document comparison works well in situations where the inputs are sequences of characters and one wants to be able to compare those sequences relative to characters, words, or lines. Unfortunately, however, 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 traditional diff tools and their input. First, comparing documents in terms of low-level documents (e.g. lines) may not result in changes that are meaningful to the domain since lines are often an artifact of presentation. For example, one can generate 'noisy diffs' by just changing whitespace. Second, the manner in which one define 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 is lost from a purely line-based approach. A similar demand is seen within other communities; scholars may want to analyze and compare texts relative to other structures such as paragraphs or sections.

1.2 Music Information Requires a Hierarchic Diff

As alluded to above, these problems are of specific importance to those in music, both in terms of usability as well as how one may want to define 'meaning' and 'similarity' among musical documents. 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 workarounds they require, the low adoption rate of computer-driven music analytic tools, and the general lack of comfort among music scholars 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. Therefore, a one-dimensional sequence of characters (the data structure for which traditional diff was designed) will not allow musicians to compare two interpretations of music. Instead, musicians need the ability to compare music and its interpretation which may be expressed hierarchically. Therefore, musicians need the ability to compare two versions of hierarchical structures. Finally, musicians may want to compare two versions of a composition, 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 composition in terms of pitch class alone, or of higher-level features like phrase structure. Moreover, a musician may want to filter a composition to compare the portions intended for a given musician or instrument type.

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. The edit distance between two trees is the minimum number of edit operatiosn necessary to transform one tree to another. The edit operations that we consider consist of

deleting, changing, and appending tree nodes. As before, a sequence of edit operations between two trees is called an edit script.

Our proposed approach leverages previous work from other domains both in industry and in academia. Within industry, there are proprietary difference engines, such as the SmartDifferencer by Semantic Designs, to compare source code in a variety of edit operations [38]. Within academia, the tree diffing problem has been long studied by theoretical computer science [9]. Different algorithms such as subtree hashing and even using IDs to align subtrees before computing similarity have been studied by researchers such as Chawathe et al. and Cobena et al.[23,30]. We employ the Zhang and Shasha tree difference algorithm to solve the edit distance between trees [151].

One benefit of this approach is that practitioners can choose a preferred level of abstraction (as defined by level within the tree) with which to summarize a change and define 'similarity'. As such, we hypothesize that XUDiff, when applied to interpretations of music as hierarchically encoded by the MEI object model, will be able to address the problems mentioned above: comparing compositions in terms of their meaning as expressed via a hierarchical interpretation of the text and thereby reducing noisy comparisons.

2.2 Suitability of MEI as a Music Representation for Collaborative Editing

MEI's tree structure affords heirarchic diff comparison. *this is where we can summarize some of the diffable features of MEI, with examples of the elements' hierarchical structure, such as...* MEI does not entangle an event series representation of musical information (MIDI) with a tree representation, as MusicXML does.

2.3 A Comparative Example

The following example demonstrates the same diff with two algorithms: first, the noisy, Least Common Subsequence algorithm, and then with the proposed hierarchic diff algorithm. *(Narrate example in detail with diagrams of both algorithms here.)*

3. CITATIONS

To distribute: [1, 2, 3, 4, 5, 6, 7, 8, 9]

4. CONCLUSIONS

The commercial presentation of widely used digital engraving tools conflates the act of sharing with act of collaboration, although these remain distinct from one another – 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." While file exchange between musical authors remains a crucial part of musical creativity, it is time for engraving software to embrace the potentials of genuinely collaborative music document editing interfaces.

In this light, the widespread publication and adoption of an effective hierarchic diff algorithm for collaborative music document editing may have a potentially far-reaching impact on many music document contexts, as it can newly enable collaborative musicology, composition, and music education. To explore these potential applications, the authors will continue to implement the described algorithm as part of the nCoda project, an open-source interface for collaborative MEI document editing. The algorithm's integration into this project also invites the design of novel graphical user interfaces and diff representation that may take into account the user's music literacy.

Acknowledgments

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

5. REFERENCES

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

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] P. Roland, "The Music Encoding Initiative (MEI)," in *Proceedings of the First International Conference on Musical Applications Using XML*, 2002, pp. 55–59.

[8] 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.

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