Renotation from Optical Music Recognition

Liang Chen, Rong Jin, Christopher Raphael

School of Informatics and Computing, Indiana University, Bloomington, IN, 47408, USA

Abstract. We describe the music renotation problem, in which one transforms a collection of recognized music notation primitives (e.g. note heads, stems, beams, flags, clefs, accidentals, etc.) into a different notation format, such as transposing the notation or displaying it in a rectangle or arbitrary size. We represent a limited degree of image understanding through a graph that connects pairs of symbols sharing layout constraints that must be respected during renotation. The layout problem is then formulated as the optimization of a convex objective function expressed as a sum of penalty terms, one for each edge in the graph. We demonstrate results by generating transposed parts from a recognized full score.

Keywords: music renotation, optical music recognition, music notation layout

1 Introduction

Optical Music Recognition holds great promise for producing the symbolic music libraries that will usher music into the 21st century, allowing flexible display, automated transformation, search, alignment with audio, and many other exciting possibilities. Work in this area dates back to the 1960s [1–11], with a nice overview

of important activity given by [12]. However, collective efforts have not yet produced systems ready for the grand challenge of creating large-scale definitive music libraries [13, 15]. Viro's recent work on the IMSLP constitutes a promising current approach to large-scale OMR [14]. Simply put, the problem is very hard, lacking obvious recognition paradigms and performance metrics [13], [27], while containing a thicket of special notational cases that make OMR difficult to formulate in a general and useful manner.

One of the many challenges, and one central to the effort discussed here, concerns the *representation* of OMR output. There are a number of fairly general music representations [16], such as MEI [17] and MusicXML [18], that provide sufficiently expressive formats for music encoding. As these representations are extensible, they can be modified to include additional information relevant for a particular perspective. However, there remains a significant gap between the natural results of OMR, which tend toward loosely structured collections of symbols, and the necessary symbol interpretations for encoding music data in these formats. Rhythm and voicing are among the issues that pose the greatest difficulty. Lurking in the background is the fact that OMR must try to capture the literal contents of the printed page, while music representations tend to take a more abstract view of music content. For instance, symbols or text that span several staves must be recognized as such in OMR, while this layout problem is not relevant for a symbolic encoding.

Rather than solving the OMR-to-encoding problem, here we explore the possibility of making OMR results useful with only a minimal understanding of the recognized symbols' meaning. In particular we address the *renotation* problem — perhaps the most important application of OMR. Renotation refers to a collection of problems that transform recognized notation into related formats, such as transposition, score-to-parts, and reformatting for arbitrary display sizes. In all cases, the output is music notation in an image format. In our approach, we seek only the level of notational understanding necessary to accomplish this task, which is considerably less than what is required by familiar symbolic representations. For instance, we do not need to understand the rhythmic meaning of the symbols. We bind our process to the original image, rather than a more abstract symbolic representation, in an attempt to leverage the many intelligent choices that were made during is construction. For instance we beam notes, choose stem and slur directions, etc. exactly as done in the original document, while we use the original symbol spacing as a guide to resolving symbol conflicts in our renotated document. However, we cannot renotate simply by cutting and pasting measures as they appear in the original source. For instance, the spacing considerations of parts are quite different than those for a score, where alignment of coincident events is crucial.

Our essential approach is to cast the renotation problem as an optimization expressed in terms of a graph that connects interdependent symbols. This view is much like the spring embedding approach to graph layout, so popular in recent years [19–22]. Our problem differs from generic graph layout in that many aspects of our desired layout are constrained, such as the vertical position of note heads, clefs and accidentals. This leads to an easier optimization task, where one can model with a convex objective functions whose global optimum is easy to identify and satisfies our layout objectives. A significant difference between our problem and graph layout is the origin of our graph edges, which come from known semantic relations between symbols. For instance we know that an accidental belongs to a note head, and thus introduce an edge that constrains their relative positions. Thus the edges of our graph come mostly from the *meaning* of the symbol, rather than their spatial proximity. This work bears some resemblance to the work of Renz [28], who takes a spring embedding view of one-dimensional music layout.

We will describe our basic approach and present a score-to-parts with example with transposition, using the *Nottorno* from Borodin's Second String Quartet.

1.1 Background: The Ceres System

Our *Ceres* OMR system [23–25] is named after the Roman goddess of the harvest, as we seek to harvest symbolic music libraries from the tens of thousands of images on the International Music Score Library Project (IMSLP) [26]. At present, *Ceres* is composed of two phases: recognition and correction. The recognition phase begins by identifying staff lines, then simultaneously grouping them into systems and identifying system measures. In the heart of the recognition process we identify the contents of each measure. Here we recognize both *composite* and *isolated* symbols. By composite symbols, we mean beamed groups and chords (including isolated notes), which are composed by grammatically-constrained configurations of note heads, stems, beams, flags, and ledger lines, as well as the decorations that belong note heads and stems (accidentals, articulations, augmentation dots, etc.) The isolated symbols we currently seek include rests, clefs, slurs, "hairpin" crescendos, text dynamics, and various other symbols. We refer to both the isolated symbols and the building blocks of the composite symbols as *primitives*.

While OMR system comparisons are suspect due to the lack of accepted metrics and ground-truthed test data [13], [27], *Ceres'* performance was competitive with what many regard as the currently-best system, *SharpEye* [1], a commercial system, in a recent limited test [23]. That said, all OMR researchers we know ackowledge that system performance varies greatly between music documents, while our evaluation was narrow relying on hand-marked ground truth. While we expect that considerable progress is still possible with the core recognition engine, it seems futile to pose the problem entirely in terms of recognition. Music notation contains a long and heavy tail of special cases and exceptions to general rules that must be handled somehow. The OMR researcher is continually faced with modeling scenarios where accounting for greater notational generality may lead to worse overall recognition performance. As an example, beamed groups can span several staves, several measures, and can have stems that go in both directions, however, allowing for these unusual cases inevitably results in false positive detections. The only reasonable modeling strategy chooses tradeoffs on the basis of overall recognition performance. Consequently, we should expect that our recognition results will contain errors, perhaps many.

Ceres complements its recognition engine with a user interface allowing a person to correct mistakes and address special cases not handled through recognition. The *Ceres* user interface, depicted in Figure 1, facilitates a drag-and-drop process we call "tagging," in which the user views the original image with the recognized results superimposed, while editing the recognized notation at the primitive level. In choosing to operate on primitives, we disregard the grammatical structure of composite symbols recovered in the recognition phase, instead treating the results as a "bag of primitives." While we lose useful information in the process, there are significant advantages to this approach. First of all, we can present the user with a well-defined task requiring no knowledge of the inner-workings of our recognition engine: she must simply cover the image "ink" with appropriately chosen primitives. In addition, operating at the primitive level al-

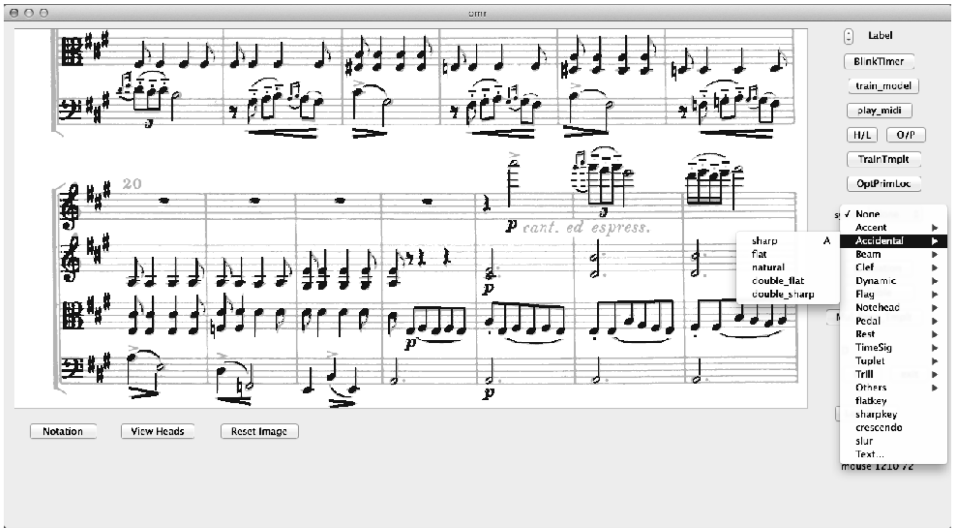

Fig. 1. *Ceres'* drag-and-drop user interface for correction primitives. Color figure with color coding of various symbol types can be seen at www.music.informatics.indiana.edu/papers/mcm15
.

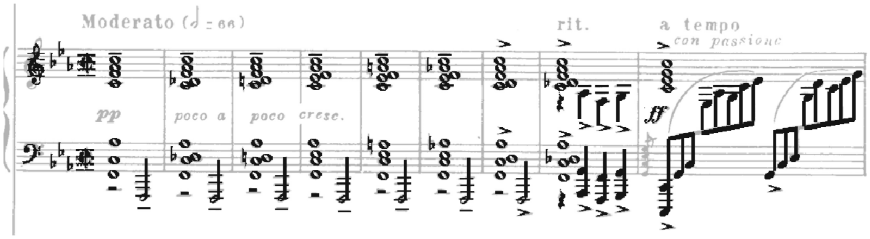

Fig. 2. Working at the primitive level allows the user to easily recover from partially correct results, as depicted in the rightmost measure. Color figure showing recognized symbols can be seen at www.music.informatics.indiana.edu/papers/mcm15
.

lows one to recover from partially correct recognition results that are awkward to handle at the composite symbol level. For instance, consider the situation of the last measure in Figure 2, where the beamed groups have each been recognized as two separate beamed groups with opposite stem directions. Converting this result at the beamed group level requires that all of the hierarchical relations are correctly expressed through the editor; in contrast, editing at the primitive level only requires that one delete the two recognized beams while substituting a longer beam in their place.

2 Music Renotation

Music *renotation* seeks to transform recognized music notation into related formats, such as converting a score into parts, transposition, or reformatting notation to fill a window of arbitrary size. Each of these tasks requires *some* degree of understanding of the notated symbols' meaning. For instance, we must know which configurations of primitives constitute beamed groups and chords, since these must be rendered subject to collective constraints (stems must terminate at beams, note heads and flags must touch stems, multiple beams must be parallel, etc.). We also need to know which symbols *belong* to which measures, since they must "travel" with the measure as the notation is reformatted. We need to understand other ownership notions such as the way accidentals, augmentation dots, and articulations belong to note heads, as spacing and alignment constraints must implicitly represent this ownership. We need to understand time coincidence between notes, rests, and other symbols, as such relations are expressed, notationally, as vertical alignment. Finally, for polyphonic music it is helpful to understand some degree of voicing as the events within a voice require spacing that makes the results readable. It is worth noting that there is much that we do *not* need to understand, including details of rhythm, meaning of most text, dynamics, and other "mark up" symbols. In fact, when transposition is not involved we don't even need to understand the pitches of the notes. As the interpretation of music notation is a challenging problem, a basic tenet of our approach here is to accomplish renotation with the minimal degree of understanding possible.

2.1 The Symbol Graph

A simple way to represent these relations is by introducing a graph structure with the *connected* symbols as vertices, meaning those collections of primitives, including singletons, that form connected regions of image ink (e.g. beamed groups or chords without their non-touching *satellites,* or any other isolated symbol). We then connect these vertices with labeled edges representing the desired spatial relationships between the symbols they connect. Thus, the graph structure describes the pairwise interrelations between symbols needed to constrain the renotation process' modification of symbol locations and parameters. We briefly describe the construction of this graph, accomplished bottom up using the known locations and labels of the symbols through a process called *assembly*. In this process we first construct the subgraphs for each composite symbol, treating the connected portion as a single symbol, and connecting the *satellites* (accidentals, augmentation dots, articulations, etc.) to the body with edges. We then create the symbol graph on the entire collection of symbols by introducing additional edges. The edges we introduce are labeled as either "horizontal" or "vertical" meaning that they constraing the horizontal or verticial distance between symbols while being indifferent to the other dimension.

We assemble the primitives into beamed groups and isolated chords in a rule-based greedy manner. In essence, we regard the primitives as being either "plugs" or "sockets," and seek to hook up these connectors, choosing an order of decisions that avoids ambiguity. For instance, every non-whole note head must connect to a stem; every beam must connect to two stems; every flag must connect to a stem, etc. For brevity's sake we will omit the individual assembly steps, as the details are many. While such approaches tend to be vulnerable to early incorrect decisions, we have not encountered this difficulty in practice. It is worth noting that the recognition and tagging processes distinguish between various "look-alike" symbols, such as *staccato* marks and augmentation dots, or key signature accidentals and note modifier accidentals, thus simplifying the process.

Occasionally during this process we encounter "plugs" with no available "socket." Such cases are nearly always due to either mislabeled or unlabeled primitives. When this occurs, our interface displays the offending composite symbol in a special color to alert the user to the inconsistency. Thus, during the tagging process, the user is directed toward uninterpretable situations requiring further attention.

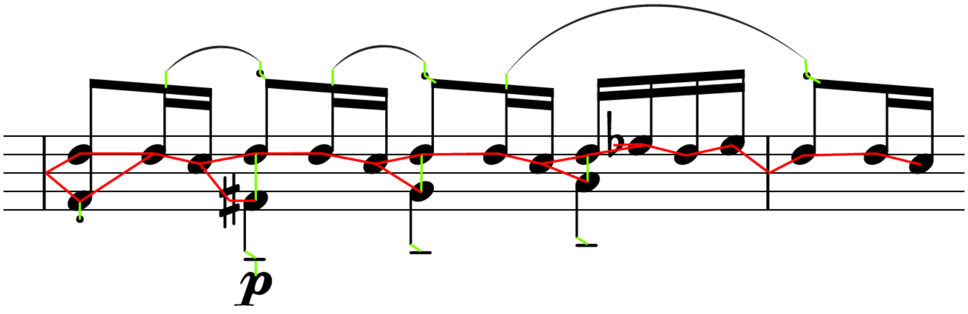

Fig. 3. Example of a symbol graph resulting from our assembly process.

Having constructed the composite symbols and connected their various satellites with edges, we then order the notes, rests, and clefs, within each staff, establishing horizontal connections between neighboring symbols, while adding vertical connections between apparently time-coincident members of this set. Clearly this process could benefit from a rhythmic or voice analysis, and we expect this will be necessary as we examine greater degrees of polyphony than exhibited in our current experiments. However, at present, we detect coincidence simply by thresholding differences in horizontal position. Finally vertical connections are established between the slurs, hairpins, and remaining symbols representing the appropriate coincidence relations.

We emerge from this assembly/tagging process with a graph structure that expresses the various ownership and hierarchical relations necessary to correctly interpret and use the symbols for renotation. Figure 3 gives an example of a symbol graph.

2.2 Music Renotation as Optimization

We formulate renotation as an optimization problem using the graph structure of the previous section. Here we denote the vertices of our graph by S (the connected symbols), while the drawing of each $s \in S$ is governed by a parameter vector $\theta(s)$. For instance, if s is a beamed group containing n notes, then $\theta(s)$ would represent the $(n+2)$-tuple needed to fully specify its rendering: the horizontal positions of the note heads and the two "corners" of the primary beam. (The vertical positions of the note are fixed at the appropriate staff position). For any non-composite $s \in S$, $\theta(s)$ simply gives the location of the symbol. In many cases there is a sharing or "tying" of parameters, representing hard layout constraints. For instance, the vertical position of an accidental, s, should be the same as the note head it modifies, thus constraining its vertical coordinate. Thus the edge between a note head and its accidental refers to the flexible horizontal distance. Similarly, a *staccato* dot should be centered above the note head it belongs to, thereby constraining its horizontal coordinate. In such cases $\theta(s)$ would have only a single component representing the "free" parameters that are not determined by such constraints.

We denote by E the collection of edges in our symbol graph. Each edge, $e = (s_1, s_2) \in E$, $s_1, s_2 \in S$, has an associated affine function that reduces the edge parameters to a single quantity: $\lambda(e) = l_1^t \theta(s_1) + l_2^t \theta(s_2) + c$. In most cases the l_1^t, l_2^t vectors simply "choose" a single parameter of $\theta(s_1)$ or $\theta(s_2)$ though we occasionlly need the greater degree of generality the linear combination offers. We then write our objective function, H, as

$$H = \sum_{e \in E} Q_e(\lambda(e)) \tag{1}$$

where Q_e is a quadratic spline function as that depicted in Figure 4.

The idea here is best illustrated in terms of an example. Consider the case of an accidental that modifies a note head. There is a desired horizontal distance

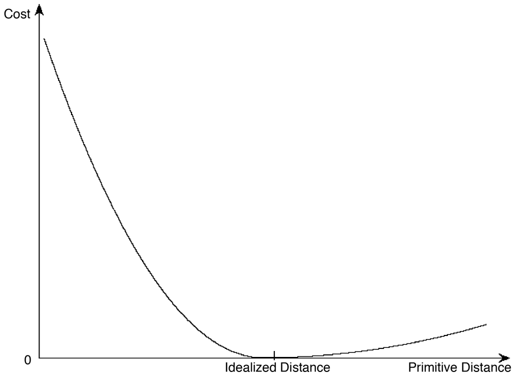

Fig. 4. The quadratic spline we use to represent the asymmetric penalty associated with an edge.

between the two: other considerations aside we would prefer to have a fixed and known distance separating the two symbols. However we feel differently about moving the two symbols closer together and further apart. As the symbols become too close they crowd one another, and eventually touch, which is highly undesirable in music layout. However, the presence of other accidentals may require that we separate the symbols further than the ideal. As this is common with the notation of accidentals surrounding a chord, this choice should not come at a great cost. Thus when $\lambda(e) < 0$, $|\lambda(e)|$ measures the degree to which we are less than the ideal, while when $\lambda(e) > 0$, $\lambda(e)$ measures the amount we are greater than the ideal. Q_e captures our asymmetric penalty for these two situations. The situation described above applies more generally to the layout problem, thus all aspects of notation we wish to control are expressed as edges in the the graph, with corresponding penalty terms.

As a sum of nearly quadratic terms, the objective function, H, is easy to optimize and converges in just a few iterations of Newton's method. As H is strictly convex, this point of convergence is the global optimum. However, since the objective function modifies the parametrization of the notation, pairs of symbols can come into contact that have no edges penalizing their spacial relations — we don't know what symbols are in potential conflict when we construct the original graph. Thus, when such conflicts arise, we augment our graph with additional edges for each conflicting pair. The penalty term for each such edge causes the symbols to "repel" thus resolving the conflict. We iterate between optimizing our objective function and modifying the function to include new terms arising from newly-detected conflicts.

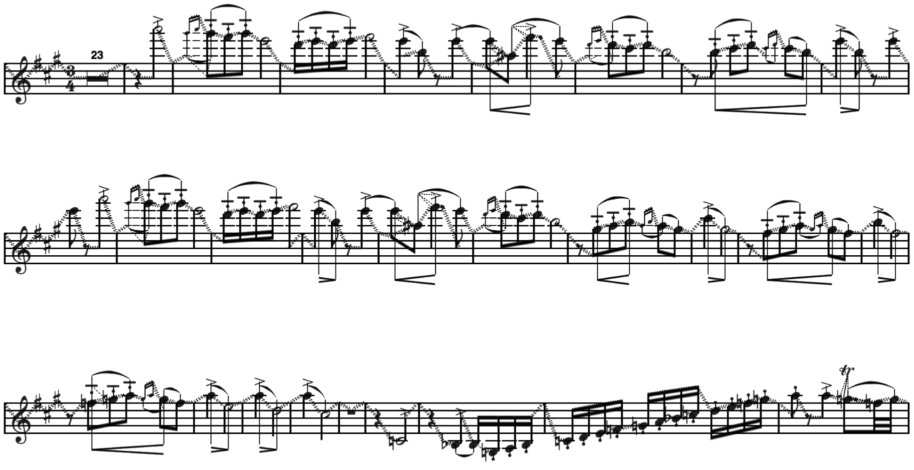

Fig. 5. A portion of the symbol graph generated for the 1st violin part of the Borodin 2nd String Quartet, 3rd movement. The red and green edges correspond to horizontal and vertical soft constraints (terms in Eqn. 1). The blue edges are terms that appeared during the optimization due to unanticipated conflicts.

3 Results and Discussion

We applied our approach in a score-to-parts application on the 3rd movement, Notturno, of Borodin's Second String Quartet, with the Ernst Eulenburg edition from around 1920. In addition, the parts were trasposed from A major to B♭ major. The complete score can be seen at `http://imslp.org/wiki/String_Quartet_No.2_(Borodin,_Aleksandr)`. In rendering our results we used the Bravura music font by Daniel Spreadbury available under the SIL open font license [30]. Perhaps the most challenging aspect of the score-to-parts problem is that scores require the alignment of time-coincident notes and rests between parts, thus creating spacing that would be completely unnatural for a single part. Thus the spacing needs to be modified considerably to create something that looks acceptable — this is essentially the main task of our algorithm.

There are some aspects of the score-to-parts problem that don't occur in our particular example, thus not addressed here. For instance, some scores, such as those for large ensemble, leave out staves of instruments that don't play on any given page. Thus the system must determine which instruments play which staves before staves can be assembled into complete parts. Also, sometimes scores will notate two or more instruments on a single staff, for instance, the bassoons and contrabassoon in orchestral score. Producing single-instrument parts in this case requires that the voices must be identified and separated. Both of these problems are examples of interest to us, though we do not implement solutions here.

Here we present results in “page” mode, in which we show the part as a single (tall) page, using line breaks and laying out the symbols so that the right bar lines of each staff align. We don’t add page breaks, though this could be handled analogously to line breaks. We accomplish this by first partitioning the measures into lines using the standard Knuth dynamic programming approach [29]. As bar lines are treated like other stand-alone symbols, this simply amounts to fixing the horizontal location of each right bar line while optimizing over the remaining parameters. Thus each line of the page is regarded as a separate optimization problem.

A portion of the resulting symbol graph for the first violin part is shown in Figure 5, while the complete set of resulting parts and graphs are available at

`www.music.informatics.indiana.edu/papers/mcm15`.

One can observe from the discussion surrounding Eqn. 1 that our objective function, H, is a sum of one-dimensional penalty terms; these penalties consider either horizontal or vertical distances between symbols, though it is possible that both arise in some cases. In Figure 5 these are shown as red (horizontal), green (vertical), and blue (conflict) edges drawn between symbols. One will also see that there are a number of edges that connect adjacent notes in a beamed group. While these are technically edges that go from a vertex to itself, this requires no change in our formulation of the problem, reflecting a desired relation, e.g. note spacing, between the parameters of a beamed group.

We perform transposition simply by moving the staff position of each note by a fixed number of steps, changing the key signature, and respelling all accidentals, viewing them as either +1,0, or -1 modifiers. For instance, an E♯ in the key of D major moves the E up by +1, and would thus appear as F double sharp in the key of E major or B♮ in the of A♭. For the Borodin we find this creates some rather unusual spellings, such as the section rendered in B double flat major (9 flats!) at the start of the fifth line of the violin part. Of course, this would normally be spelled as A major, though this would require harmonic analysis to detect, and certainly constitutes a rare case.

A notation maven may find a fair bit to criticize about the resulting layout. We view our effort as more of a proof of concept, rather than recipe for ideal notation. We have not considered many aspects of layout usually included in serious notation systems, such as spacing that reflects note length. Rather, it has been our intent to show that OMR results can be utilized effectively by addressing only a minimal portion of the symbol interpretation problem. We continue to explore the symbol graph as a possible alternative to more expressive music representations such as MEI and MusicXML — capturing less about the notational structure and its meaning, but easier to derive automatically from OMR. Analogous approaches are promising for other OMR applications requiring symbol interpretation, such as music playback.

References

1. Jones G., Ong B., Bruno I., Ng K.: Optical Music Imaging: Music Document Digitisation, Recognition, Evaluation, and Restoration. Interactive Multimedia Music

Technologies, 50-79, IGI Global, Information Science Reference, (2008)

2. Ng K. C., Boyle R. D.: Recognition and Reconstruction of Primitives in Music Scores. Image and Vision Computing, 14(1), 39-46, (1996)
3. Bitteur H.: Audiveris. https://audiveris.kenai.com/ (2014)
4. Fujinaga I.: Adaptive optical music recognition. Ph.D. Thesis, McGill University, Montreal (1997)
5. Pugin L., Burgoyne J. A., Fujinaga I.: MAP Adaptation to Improve Optical Music Recognition of Early Music Documents Using Hidden Markov Models. Proc. International Symposium on Music Information Retrieval, 513-516, (2007)
6. Choudhury G. S., DiLauro T., Droettboom M., Fujinaga I., Harrington B., MacMillan K.: Optical Music Recognition System within a Large-Scale Digitization Project. Proc. International Symposium on Music Information Retrieval (2000)
7. Fahmy H., Blostein D.: A Graph-rewriting Paradigm for Discrete Relaxation: Application to Sheet-Music Recognition. International Journal of Pattern Recognition and Artificial Intelligence, 12(6), 763-799 (1988)
8. Rossant F., Bloch I.: Robust and Adaptive OMR System Including Fuzzy Modeling, Fusion of Musical Rules, and Possible Error Detection. EURASIP Journal on Applied Signal Processing, vol. 2007 (2007)
9. Couasnon B., Retif B: Using a Grammar for a Reliable Full Score Recognition System. Proceedings of International Computer Music Conference 187-194, (1995)
10. Carter N. P.: Conversion of the Haydn Symphonies into Electronic Form Using Automatic Score Recognition: A Pilot Study. Proceedings of SPIE, 279-90 (1994)
11. Bainbridge D., Bell T.: The Challenge of Optical Music Recognition. Computers and the Humanities, vol. 35, 95-121, (2001)
12. Fujinaga I.: Optical Music Recognition Bibliography. http://www.music.mcgill.ca/~ich/research/omr/omrbib.html (2000)
13. Rebelo A., Capela G., Cardoso J. S.: Optical Recognition of Music Symbols. International Journal on Document Analysis and Recognition, vol. 13, 19-31, (2009)
14. Viro V.: Peachnote: Music score search and analysis platform. Proceedings of the International Society for Music Information Retrieval Conference (ISMIR), 359-362, (2011).
15. Blostein D., Baird H. S.: A Critical Survey of Music Image Analysis. in Structured Document Image Analysis, eds. H. S. Baird, H. Bunke and K. Yamamoto, 405-434, Springer-Verlag, Berlin, (1992)
16. Selfridge-Field E.: Beyond MIDI : The Handbook of Musical Codes. MIT Press, Cambridge (1997)
17. Hankinson A., Roland P., Fujinaga I.: The Music Encoding Initiative as a Document Encoding Framework. 12th International Society for Music Information Retrieval Conference, 293-298, (2011)
18. Good M.: MusicXML for Notation and Analysis. Computing in Musicology vol. 12, (2001)
19. Kabourov S.: Spring Embedders and Force Directed Graph Drawing Algorithms. CoRR, vol. abs/1201.3011, (2012)
20. Fruchterman T. , Reingold M.: Graph Drawing by Force-Directed Placement. Software Practice & Experience (Wiley) 21 (11): 11291164, (1991)
21. Eades P.: A heuristic for graph drawing., Congressus Numerantium 42 (11): 149160 (1984)
22. Kamada T., Kawai S.: An algorithm for drawing general undirected graphs. Information Processing Letters (Elsevier) 31 (1): 715, (1989)
23. Raphael C., Jin R.: Optical music recognition on the International Music Score Library Project. Document Recognition and Retrieval XXI, (2014)

24. Jin R., Raphael C.: Interpreting Rhythm in Optical Music Recognition. Proc. International Symposium on Music Information Retrieval, 151-156 (2012)
25. Raphael C., Wang J.: New Approaches to Optical Music Recognition. Proc. International Symposium on Music Information Retrieval, 305-310 (2011)
26. Guo E.: The IMSLP, Petrucci Music Library. `http://imslp.org/` (2014)
27. Byrd D., Schindele M.: Prospects for Improving Optical Music Recognition with Multiple Recognizers. Proc. International Symposium on Music Information Retrieval, 41-46, (2006)
28. Renz K.: Algorithms and Data Structure for a Music Notation System based on GUIDO Notation. PhD. Dissertation, Technischen Universität Darmstadt Proc. International Symposium on Music Information Retrieval, (2002)
29. Knuth D., Plass M.: Breaking Paragraphs Into Lines in Software - Practice and Experience. 11, 1119-1184, (1981)
30. Spreadbury D.: `http://www.smufl.org/fonts/`