Nineteenth Australasian Computer Science Conference

An extensible Optical Music Recognition system

David Bainbridge
Department of Computer Science
University of Waikato
Hamilton, New Zealand
d.bainbridge@cs.waikato.ac.nz

Tim Bell
Department of Computer Science
University of Canterbury
Christchurch, New Zealand
tim@cosc.canterbury.ac.nz

Abstract

Optical music recognition (OMR) is a form of structured document image analysis where symbols overlaid on the conventional five-line stave are isolated and identified so that the music can be played through a MIDI system, or edited in a music publishing system. Traditionally OMR systems have had recognition techniques hard-coded in software. This paper describes a system that has been designed to be extensible without the need to change the system's source code. Extensibility is achieved by providing tools for music recognition that are used to tailor the system to suit the type of music being recognised. The tools include a selection of methods for identifying staves and isolating objects from them, methods for describing and identifying primitive musical shapes, and a grammar for specifying the relationships between the shapes that are recognised. The system is flexible enough to work with different publishers' symbol sets, and even with different types of music notation, such as the square-note notation used in early music.

Keywords music recognition, document image analysis, two-dimensional grammars

1 Introduction

Optical Music Recognition (OMR) enables people to scan printed music onto a computer and have it played by the computer, or brought up in a music editor. OMR has many applications. For example, a soloist can practise with a computer playing an accompaniment; a musician can have a piece of music transposed to another key automatically; the notation of the music could be converted (for example, to tablature or Braille); or an old edition of the music could be re-edited.

OMR is related to Optical Character Recognition (OCR), but presents some significant extra challenges. One important different is that music is two-dimensional (pitch vertically and time horizontally), while text is essentially one-dimensional. Another significant complication in music is that the symbols are usually overlayed on a five-line stave, so it is difficult to isolate symbols. Also, unlike OCR, symbols are made up of components that can be combined in many ways. For example, a stem may hold a number of note heads, and may be beamed to another stem. A further difference is that the same symbol may appear in different forms. For example, the lengths of slurs, ties, beams, and stems will depend on the context in which they are used.

OMR is complex and must incorporate many rules about what can appear in musical notation. Existing systems tend to have the rules hard-coded into the OMR program (see Selfridge-Field [13] for a survey of recent systems). This paper describes the CANTOR system, which has been designed to be as general as possible by allowing the user to define the rules that describe the music notation. This means that the system is readily adapted to variations of notation used by various publishers for Common Music Notation (CMN), and it also means that completely different forms of notation can be recognised, such as the square-note notation used with four-line staves for early music.

Other systems have been designed for adaptation either by writing the program so that it is easily modified [6] or by keeping track of corrections made to a recognised document, and ``learning'' not to make the same mistake next time [12]. However, although these systems can incorporate new features of CMN, they are not easily adapted to alternative notations.

OMR systems generally have the following four stages in their recognition process.

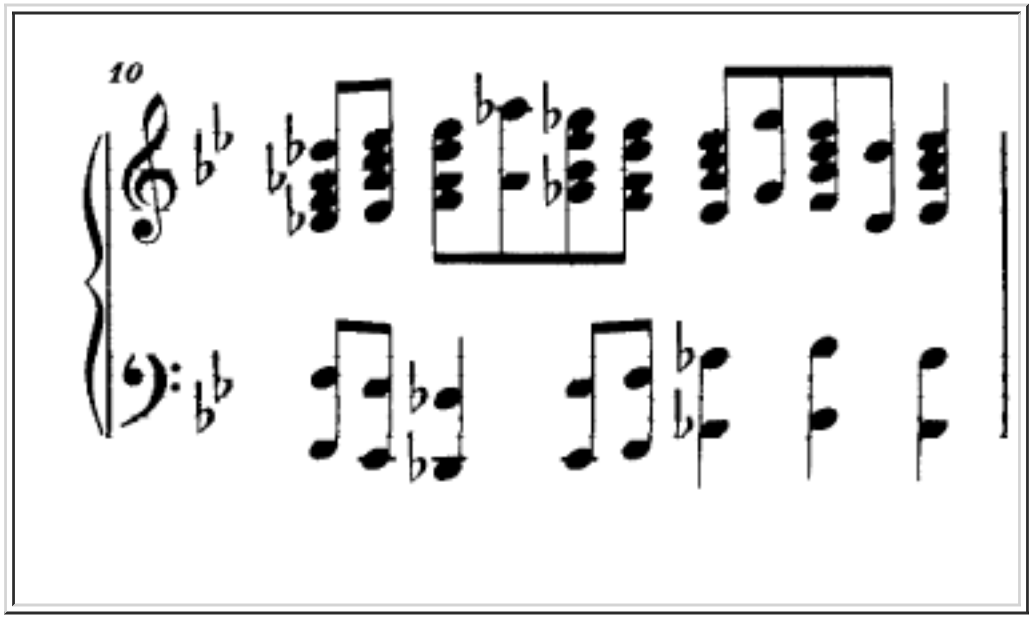
Stave line identification
: The position of the staves is identified, and (usually) the lines are removed, leaving the superimposed musical symbols (see Figure 1a).

Musical object location
: The symbols that were on and around the stave are located (Figure 1b).

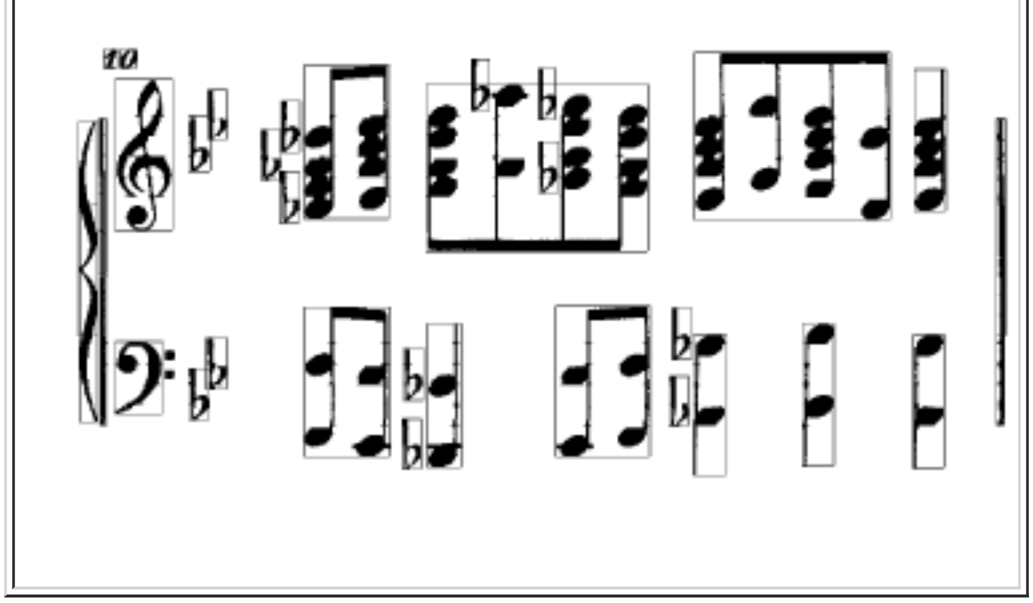
Symbol identification
: The type of each symbol is determined (Figure 1c).

Semantics of music notation
: The relationship between symbols is determined and the information is stored in a form that programs such as sequencers or music editors can use (Figure 1d).

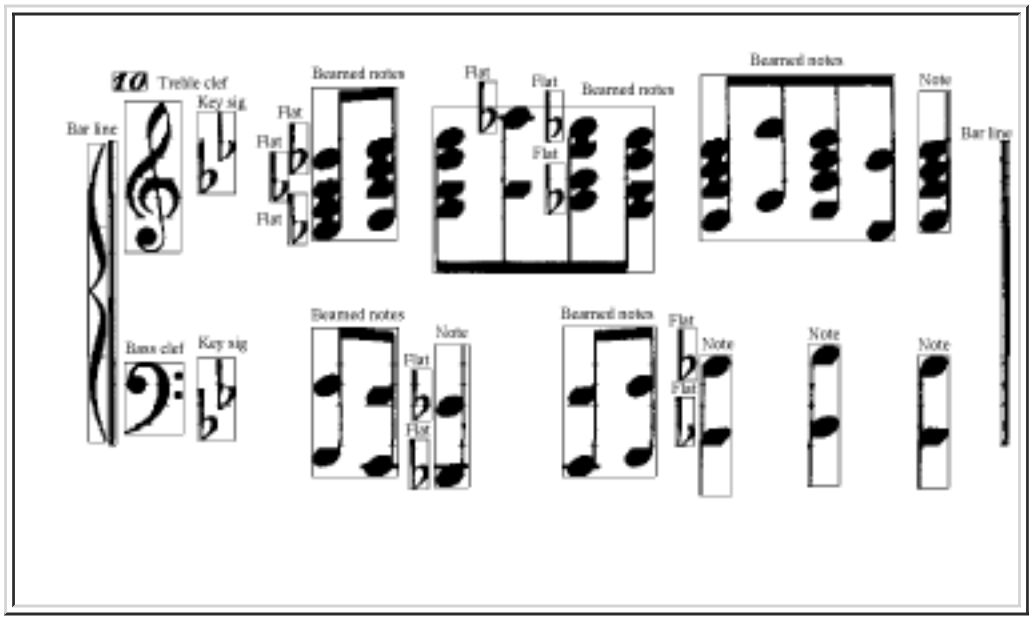
This paper focuses on the symbol identification stage, as this is the one that is the most dependent on the kind of notation being recognised. The structure of the paper is as follows. Section 2 describes the pre-processing required to isolate objects on a score. Section 3 discusses a general system that has been developed for recognising the primitive objects that have been isolated, and Section 4 describes a technique for assembling these objects into musical features. In Section 5 we discuss how the system can be adapted to alternative music notations.



(a)



(b)



(c)

```
Staff System 1:
Staff 1:
{4Db 4F 4Ab 5Db}(0.5),{4Eb 4Ab 5C 5Eb}(0.5),{4F 4Ab 5Db 5F}(0.5),{4Ab 5Ab}(0.5),
{4Gb 4Bb 5Eb 5Gb}(0.5),{4F 4Ab 5Db 5F}(0.5),{4Eb 4Ab 5C 5Eb}(0.5),
{4Gb 5Gb}(0.5),{4F 4Bb 5Db 5F}(0.5),{4Db 5Db}(0.5),{5Eb 4Eb 4Ab 5C}(1.0) |
{4Ab 3Ab}(1.0)&4Eb(3.0),{4Bb 3Bb}(1.0),{4Ab 3Ab}(1.0),{4Bb 3Bb}(0.5)&4Eb(1.0)\_,
{4C 5C}(0.5),{5Eb 4Eb}(0.5)&_/4Eb(0.5),{3Bb 4Bb}(0.5) | .....

Staff 2:
{2F 3F}(0.5),{2Eb 3Eb}(0.5),{3Db 2Db}(1.0),{2Eb 3Eb}(0.5),{2F 3F}(0.5),
{2Ab 3Ab}(1.0),{2Bb 3Bb}(1.0),{2Ab 3Ab}(1.0) | {1Gb 2Gb}(2.0),{2F 1F}(1.0),
{1Gb 2Gb}(1.0),{2Gb 3Gb}(0.5),rest(0.5) | .....
```

(d)

Figure 1: Stages in a typical OMR process (a) removal of the staves (b) locating objects (c) identifying symbols (d) determining the semantics.

2 Detecting stave lines and locating objects

The first stage in OMR is identifying where the stave lines are. This is usually straight-forward because the five lines are a very regular feature on the page, and can be found reliably by taking a horizontal projection of the number of black pixels on each line of the image. Stave lines will be marked by peaks in the histogram of the projection. Unfortunately such peaks deteriorate if the page was not straight when scanned. A simple modification to compensate for this is to restrict the horizontal projection to short sections on the left hand side and right hand side of the page, and then connect the corresponding detected lines. Identifying stave lines in this manner can be used to establish the angle of rotation required to correct the scanned image. Occasionally large horizontal objects such as phrase marks may cause ``false'' peaks, but this problem can be avoided by looking for regular spacing in the peaks.

A standard OCR method is then applied to the image at this stage to identify and remove any text (such as titles, lyrics, part names, and dynamic markings) on the image. Applying it at this stage means that the OCR package is dealing with a straightened page, and simplifies later stages because the music recognition system does not need to be concerned about misinterpreting parts of the text as a musical feature.

A new image is then constructed in which the lines are removed. Because musical features overlap the lines, it is not a good idea to simply remove all of the black pixels along a line, as considerable post-processing is needed to reconstruct objects that get broken up. Several methods have been proposed to prevent objects being broken up. The CANTOR system uses a method described by Clarke [7] in which black pixels are removed only if there are no black pixels above or below the line in the vicinity of the one under consideration. This process can still break up some musical features, but we retain the original image as well, which enables us later on in the system, to check whether a black pixel has been removed in cases where the recognition system would expect to find one.

The musical objects left after stave removal are located by searching for black pixels, and finding connected pixels using a flood-fill algorithm [10]. One method of coping with fragmented objects is to devise a heuristic that merges bounding boxes together by comparing adjacent bounding boxes in the vicinity of stave lines. For example, Figure 2a shows a bass clef that is broken up by stave removal (Figure 2b), but reconstructed by the joining of bounding boxes (Figure 2c). An alternative to this approach would be to reconstruct the bass clef at a later stage by defining a bass clef to be constructed from the two components in Figure 2b. Another possibility is to use the isolated shapes to detect where objects might be, but to match against the original image if the shape is prone to being broken up by stave-line removal. Fragmentation is a serious problem in OMR, and it is valuable to have several methods available to overcome the complications that it causes.

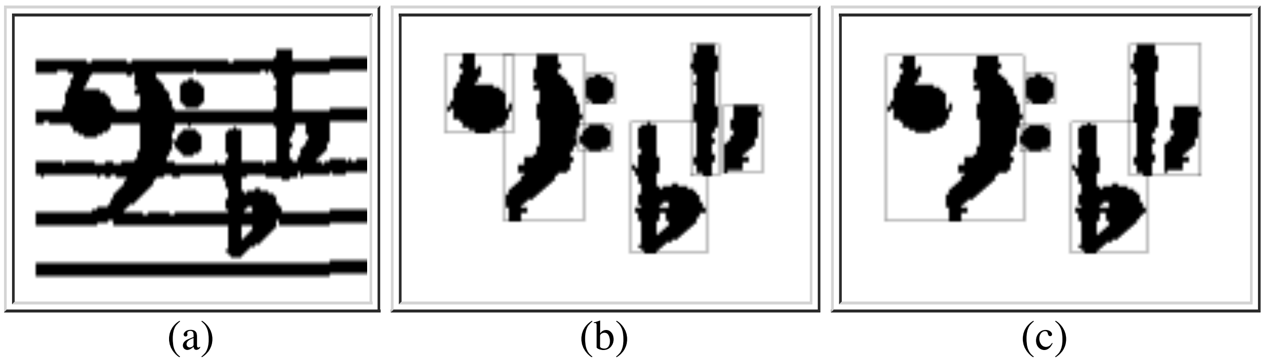
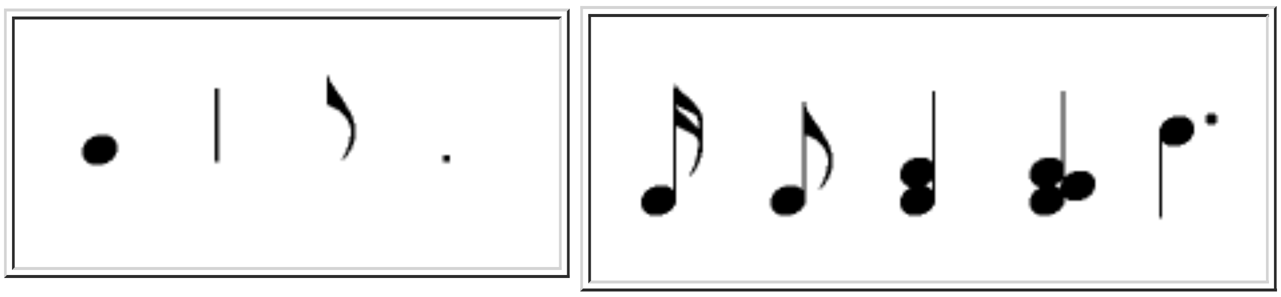


Figure 2: Locating a bass clef (a) the original image (b) broken up by stave removal (c) reconstructed.

3 Recognising primitive objects

In the next phase, CANTOR recognises the primitive objects on the page of music from which the staves have been removed. It looks for the components of musical features, such as note-heads, stems, and tails, rather than whole objects, because there are many ways of combining primitive objects to create musical features (Figure 3).



(a) (b)

Figure 3: (a) Primitive objects (b) Musical features constructed from primitive objects.

Various techniques have been used in the literature to recognise objects after the staves have been removed. A commonly used method is to use a set of templates that are matched against objects that have been isolated. A measure is made of the similarity between an object being matched, and each available template, with the highest rating match being used, provided that it passes some threshold. The drawback with this method is that the shapes of objects can vary greatly from publisher to publisher, and even within a piece of music. Also, some shapes are very similar, such as bar lines and stems, and even hollow and filled note-heads, while other musical features appear in an infinite variety of variations, such as slurs and beams. Furthermore, matching can be slow because each isolated object must be compared with every template (which can be particularly time-consuming if many variations of each symbol are stored, and symbols can be scaled.)

Several alternatives have been proposed to either accelerate this process, or to make it more reliable. Fujinaga [11] used *projections* (both horizontal and vertical) to create a signature of an object, and match that with projections of previously identified objects. The *Hough transform* is used in image processing to locate straight lines and curves [4] and has proved useful for finding bar-lines, stems, and the curl of a bass clef. The technique of *slicing* involves taking a cross-section through an object at a predetermined position, and counting the number of transitions from black to white pixels [1]. For example, a slice near the bottom of a natural sign (♮) would find one transition, while a slice across the same part of a sharp sign (♯) would find two. Like slicing, *connectivity analysis* [2] is another pattern recognition technique that is particularly suited to classifying musical features that are dynamic in their size within one piece of music, such as slurs. Connectivity analysis answers the question, is there a path through the object from (x_1, y_1) to (x_2, y_2) (where $x_1<x_2$) with x monotonically increasing. Connectivity analysis is useful for detecting beams and slurs.

Rather than opting for just one of these techniques, CANTOR makes them all available. They are invoked through a language designed as part of CANTOR for this purpose, called PRIMELA (PRIMitive Expression LAnguage), through which any combination of these techniques can be used to identify a primitive object in the input image. The system also includes a graphical editor that can be used to draw free-form shapes, which can be used as scalable templates, as profiles for projections, and to provide curves for the Hough transform. The free-form images can be based on scanned examples from music. A part of a scanned image is placed in the background of the graphical editor, and free-form curves can then be placed on top of it, capturing the main features of a shape rather than the specific features of the scanned example. The editor can also perform a projection of a shape, and a characteristic curve can be drawn for the projection rather than the original shape. The graphical editor provides textual output, which is simply a list of points on a B-spline curve that describe the object. This output is used directly in the PRIMELA language.

Figure 4 shows some PRIMELA code for recognising a treble clef. The `size` and `normalise` commands determine the size the symbol is relative to the size of a stave. The clef is matched by comparing the horizontal projections of the pre-drawn treble clef with the same projection of the object in the image that is being identified (`yproject` on line 4). The model treble-clef is described by the list of 14 points on an open B-spline curve (lines 8 to 11). This curve was created by fitting a B-spline curve onto the projection of a scanned treble clef using the graphical editor.

```
1  primitive treble_clef
2   : size(42,145), normalise(1.0)
3  {
4    yproject ptc
5      {
 6        transform: scale(0.333333,0.333333), translate(23.333333,106.333333)
 7          {
 8            bspline 2 open 14
 9                       (-70  115) (-01   95) (-02   62) (-61   22) (-48  -39)
10                       (-27 -100) ( 56 -107) ( 43 -133) (-44 -175) (-09 -217)
11                       (-65 -230) (-67 -271) (-29 -293) (-56 -319);
12          }
13      }
14
15    initialise
16      {
17        string option_box = "object";
18
19        int yproject_lower_match = 70;
20        int yproject_upper_match = 75;
21
22        int y_copy_expand  = 2;
23      }
24
25    match
26      {
27        pre       {% %}
28        rect      {% rect_yb_ += y_copy_expand;                         %}
29        copy      {% %}
30        post      {% %}
31        certainty {% linear_certainty(yproject_lower_match,score_match_,
32                                      yproject_upper_match);            %}
33        remove    {% %}
34      }
35  }
```

Figure 4: PRIMELA code for recognising a treble clef.

The `initialise` section (lines 15 to 23) sets some thresholds and declares dynamic variables for use during matching, which is then performed in the `match` section (lines 25 to 34). The `match` section describes criteria for making a match, such as pre-conditions that must be satisfied by the image, a tolerance for the bounding rectangle that contains the object, thresholds for the certainty of a match before it will be accepted, and rules for removing the recognised object from the image. By leaving the brackets empty for the copy and remove commands, a default behaviour of copying and removing (respectively) every pixel in the rectangular regions, is invoked.

Objects that are recognised with a high certainty are removed from the image, as this simplifies later searches because it reduces the set of objects to which an unknown one might belong. Objects that pass a lower matched threshold are tagged as ``uncertain'' primitives and given a rating between zero and one to reflect this, but are left in the image. The certainty rating is used later on in CANTOR, when the primitive shapes are combined into valid musical features.

PRIMELA code for CMN is currently under development, and is being tested on a number of pieces from different publishers. Table 1 shows statistics from using PRIMELA to recognise primitive objects in two pages of the piece ``Big My Secret'' by Michael Nyman, published by Chester Music. The opening of this piece is shown in Figure 7. Table 2 shows the same statistics for the piece ``Bonita'' by Vander Cook, published in 1943 by Rubank Inc. The music for this had a two-stave piano part, and a violin part on a smaller stave. Some of the musical features were touching (such as a flat superimposed on a slur), which provides an extra challenge for OMR.

Feature	Number in Original	Number recognised correctly	False hits
Treble clef	8	8	0
Bass clef	8	8	0
Sharp	32	31	0

Vertical line	495	495	9
Slur	67	64	1
Beam	199	185	0
Filled note head	490	490	0
Dot	44	44	0

Table 1: Statistics for recognition of primitives in ``Big My Secret''

Feature	Number in Original	Number recognised correctly	False hits
Treble clef	16	15	0
Bass clef	8	8	0
Crochet rest	41	40	3
Rectangular rest	24	17	0
Sharp	14	12	0
Natural	18	17	0
Flat	68	44	2
Vertical line	379	379	0
Slur	34	12	2
Beam	33	32	1
Filled note head	476	476	21
Dot	43	16	0

Table 2: Statistics for recognition of primitives in ``Bonita''

The tables show high recognition rates for many objects, and the poor performance on some objects was mainly due to weaknesses in the PRIMELA descriptions, which are still under development. Our experience developing these descriptions indicates that it is not difficult to refine the code to eliminate false matches and unrecognised symbols.

The nine false hits for vertical lines in Table 2 are due to mistaking tails on quavers, and the number ``1'' for a vertical line. This could be avoided by searching for quaver tails before vertical lines, and by using a better OCR system to remove the numbers more reliably. The 14 beams not recognised were shorter than the minimum length in the PRIMELA description.

A number of errors occurred recognising the ``Bonita'' piece because the PRIMELA description is yet to be fine tuned for it. A large number of slurs were missed because they are thick in this piece, and were confused with beams. This could be fixed by making the recognition rely more on the curvature of the object than its thickness. The large number of false hits for note-heads was mainly due to a beam being mistaken for a series of note-heads. A large number of dots were missed simply because the precondition for size (designed for another publisher's style) was too small.

4 Assembling primitive objects into musical features

Once the primitive objects on the page have been identified, they must be assembled into the larger objects that they are a part of. For example, note-heads and stems combine to form notes, and a sequence of sharps in a particular context can form a key signature. Such assembly tasks are common in structured document image analysis problems, and a variety of techniques exist [3]. A grammar-based approach for music recognition is favoured since it also meets our specific requirement of generality--each form of notation would have its own grammar that defines valid configurations of the primitive shapes.

The assembly phase in CANTOR is implemented using Definite Clause Grammars (DCG's). DCG's are similar to BNF, but use a slightly different notation which make them amenable to implementation in a Prolog environment. Many Prolog systems have facilities for processing DCG's [5]. In Prolog, a DCG can be used to construct sentences in the grammar as well as parse existing sentences. This flexibility is ideal for assembling musical objects.

Music notation is two-dimensional, so some adaptation needs to be made to conventional grammars to allow for this. The OMR system devised by Coüasnon *et al*. [8, 9] is also based on a DCG. They cope with two-dimensions by extending the set operators that the grammar can use, and relying on λProlog (a higher-order extension of Prolog). However, they report that developing a parser is a difficult task. This is due to the more complex task that their grammar is expected to perform, as it forms the top level control to their system, and not only assembles primitive shapes into musical features, but also controls when to join and segment shapes, as well as specifying ``how to read music.'' No statistical results are available for their system.

Another option is to replace the one-dimensional list of tokens with a higher order data-structure. Tree grammars and graph grammars use this idea, and have been employed successfully in two-dimensional document image analysis. A drawback of this approach is that the increased generality increases the complexity of the parser.

CANTOR avoids such complications by using the conventional form of a DCG, except instead of using a *list* of tokens for the input, a *bag* of tokens is used. Instead of getting a unique next symbol, the grammar can ``request'' a token (say a note-head) from the bag, and if its position does not fit in with the current musical feature (say a note) that is being parsed, then the grammar can back-track and request the ``next'' note-head from the bag. Constraints such as ``must be close to the selected stem'' are naturally incorporated into a DCG by embedding Prolog into it. It is relatively simple to add bags to a DCG; for our system a 247 line implementation of a DCG increased in size by 90 lines to accommodate bags. Also, some additional terms were added, such as `prim_present` (primitive present), to provide additional information for assembling the primitives.

Figure 5 shows parts of the DCG for describing valid configurations for a simple ``up'' note, that is, one or more note-heads with a stem going up. The first production chooses a vertical line and looks for tails or note-heads that are close by. The second production allows multiple note-heads to be associated with the same stem. The `note_head_within_up` production checks to see if a selected note-head is close enough to the stem. The `note_head` production selects note-heads to check in the `note_head_within_up` production. It also discovers any duration modifying dots to the right of the note-head using `opt_dur_dots`, and stores the duration of the note in the derivation tree.

```
basic_note_up
         ==> [(vertical_line,_,Sxl,Syt,Sxr,Syb,_)],
             opt_tails_up(Sxl,Syt,Syb,NoTails),
             note_heads_up(Sxl,Syt,Syb,NoTails).

note_heads_up(Sxl,Syt,Syb,NoHalves)
         ==> note_head_within_up(Sxl,Syt,Syb,NoHalves),
             note_heads_up(Sxl,Syt,Syb,NoHalves).
note_heads_up(Sxl,Syt,Syb,NoHalves)
         ==> note_head_within_up(Sxl,Syt,Syb,NoHalves).

note_head_within_up(Sxl,Syt,Syb,NoHalves)
         ==> note_head(Syt,Syb,NoHalves,Nxl,Nyt,Nxr,Nyb),
             { % note head must be close to stem (to the left) }.
```

```
note_head(Syt,Syb,NoHalves)
         ==> [(full_note_head,_,Nxl,Nyt,Nxr,Nyb,_)],
             { % retrieve appropriate StaffGap for note head },
               opt_dur_dots(Nxr,StaffGap,Nyt,Nyb,NoDots),
             { % store note head duration }.

opt_dur_dots(Nxr,StaffGap,Nyt,Nyb,NoDots)
         ==> opt_dur_dots(Nxr,StaffGap,Nyt,Nyb,0,NoDots).
opt_dur_dots(Nxr,StaffGap,Nyt,Nyb,NoDotsIn,NoDotsOut)
         ==> [(dot,_,Dxl,Dyt,Dxr,Dyb,_)],
             { % check dot close enough to note head },
             { IncNoDots is NoDotsIn + 1 },
               opt_dur_dots(Dxr,StaffGap,Nyt,Nyb,IncNoDots,NoDotsOut).
opt_dur_dots(_,_,_,_,NoDots,NoDots)
         ==> []. % epsilon

% opt_tails_up similar to opt_dur_dots
```

Figure 5: DCG clauses that describe a simple ``up'' note consisting of note heads, a stem, dots and tails.

The constraints used in the DCG that measure the ``closeness'' of objects are normalised with respect to the size of the stave that they are on. A score can have different size staves for different parts, so some care is required to identify which stave a feature belongs to. A note is usually associated with the stave it is nearest to. For notes that are close to two different-sized staves we perform a match at both sizes, and use the best match. More accuracy could be obtained by locating ledger lines and using these to determine which stave a note belongs to.

Because the parser is based on back-tracking, the amount of time taken can increase combinatorially. Prudent use of the cut operator is necessary to achieve tractable time complexities.

Attributed derivation trees of the assembled musical features are constructed during the parse, from which the semantics of the music can be determined. A derivation tree for a crochet note is shown in Figure 6. Determining the semantics is also partially done during parsing--for example, while parsing a note it is possible to calculate its duration, and this information can be conveniently passed on to the musical semantics stage by storing it as an attribute at the appropriate node of the derivation tree.

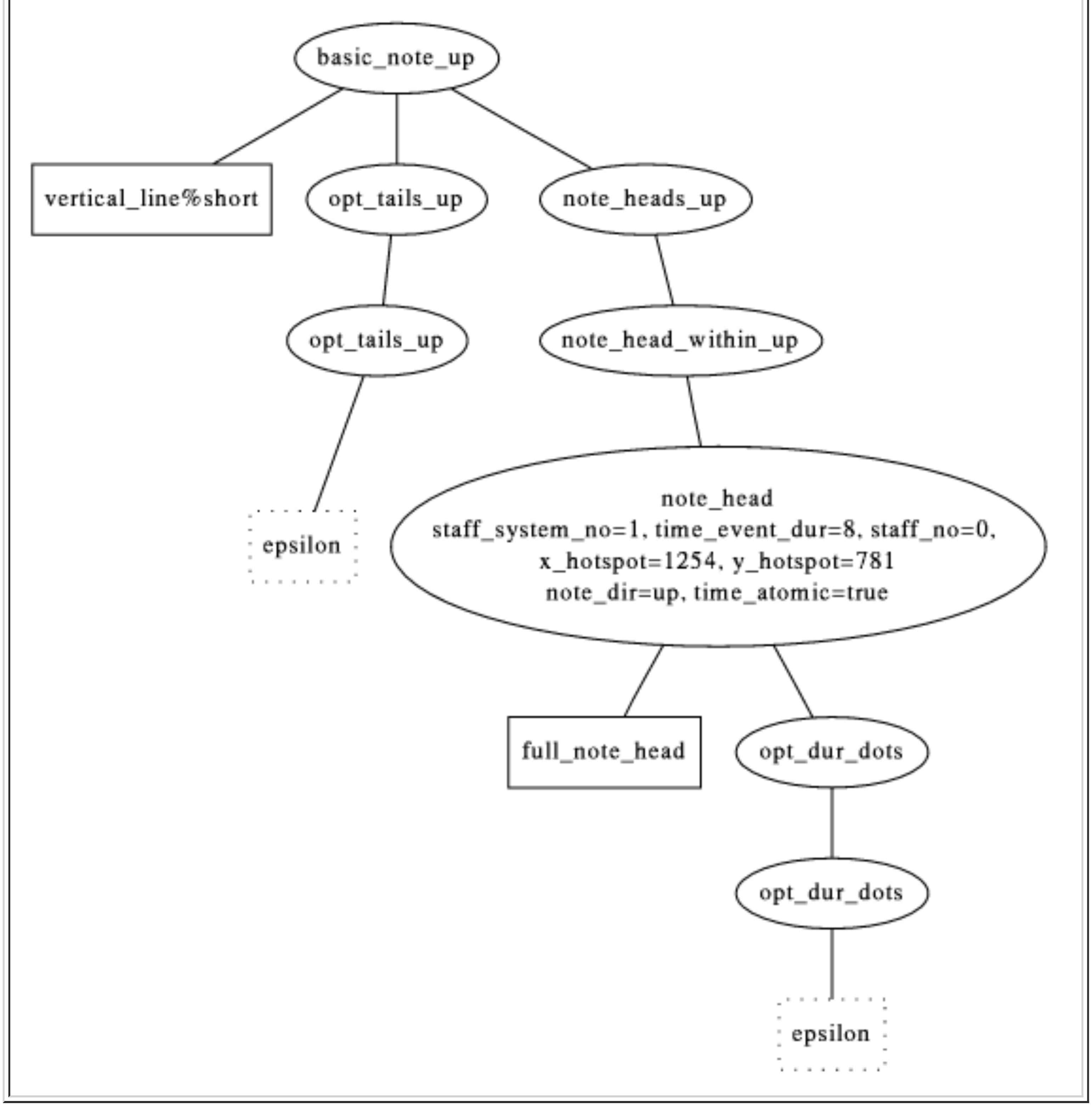


Figure 6: A derivation tree for crotchet note.

Table 3 shows statistics for the assembly of components recognised by PRIMELA for ``Big My Secret.'' Table 4 shows the same statistics for ``Bonita.''

Components assembled	Number required	Number assembled correctly	Incorrect associations
Dot with note-head	22	22	0
Note-head with stem	483	483	0
Note with beam	411	410	0
Nested beams	99	99	0
Bass clef curl with two dots	8	8	0
Bar-line on stave	25	25	1

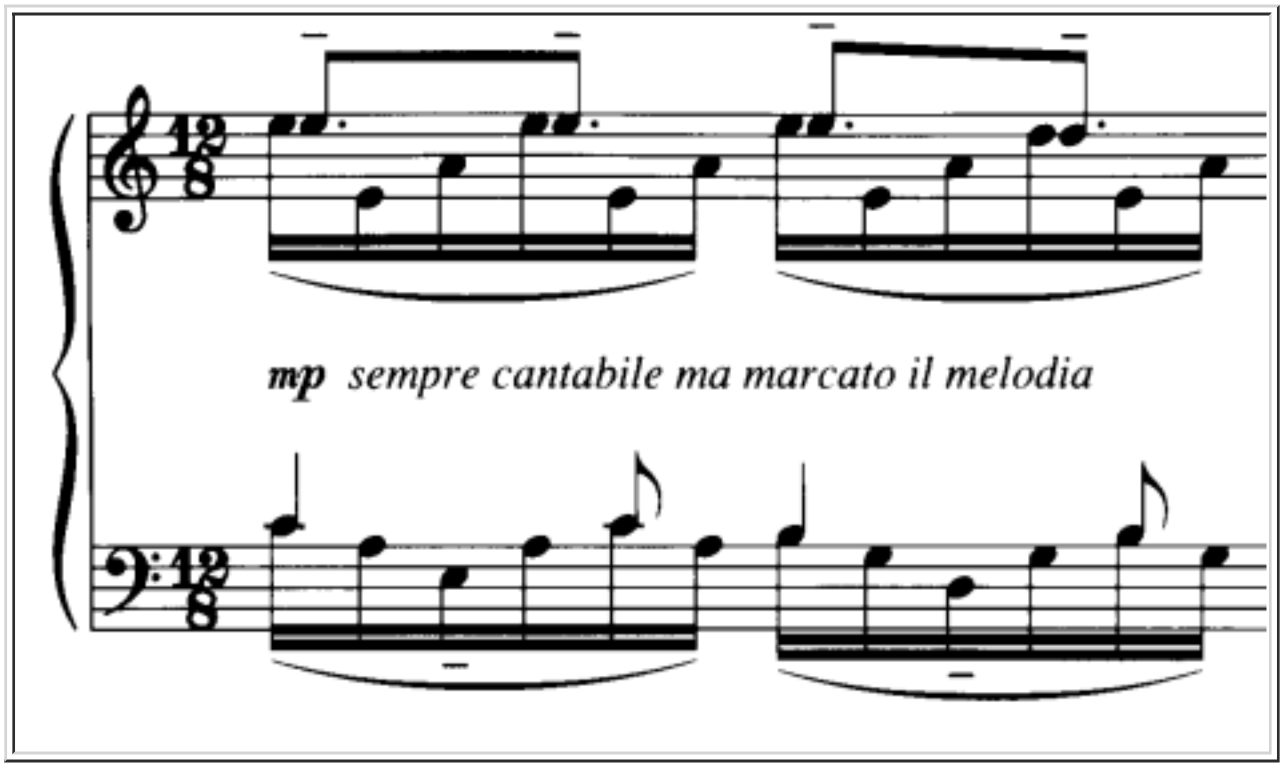
Table 3: Statistics for assembly of primitives in ``Big My Secret''

Components assembled	Number required	Number assembled correctly	Incorrect associations
Dot with note-head	6	6	0
Note-head with stem	478	478	0
Note with beam	88	88	0
Nested beams	4	4	0
Bass clef curl with two dots	2	2	0
Bar-line on stave	90	90	1

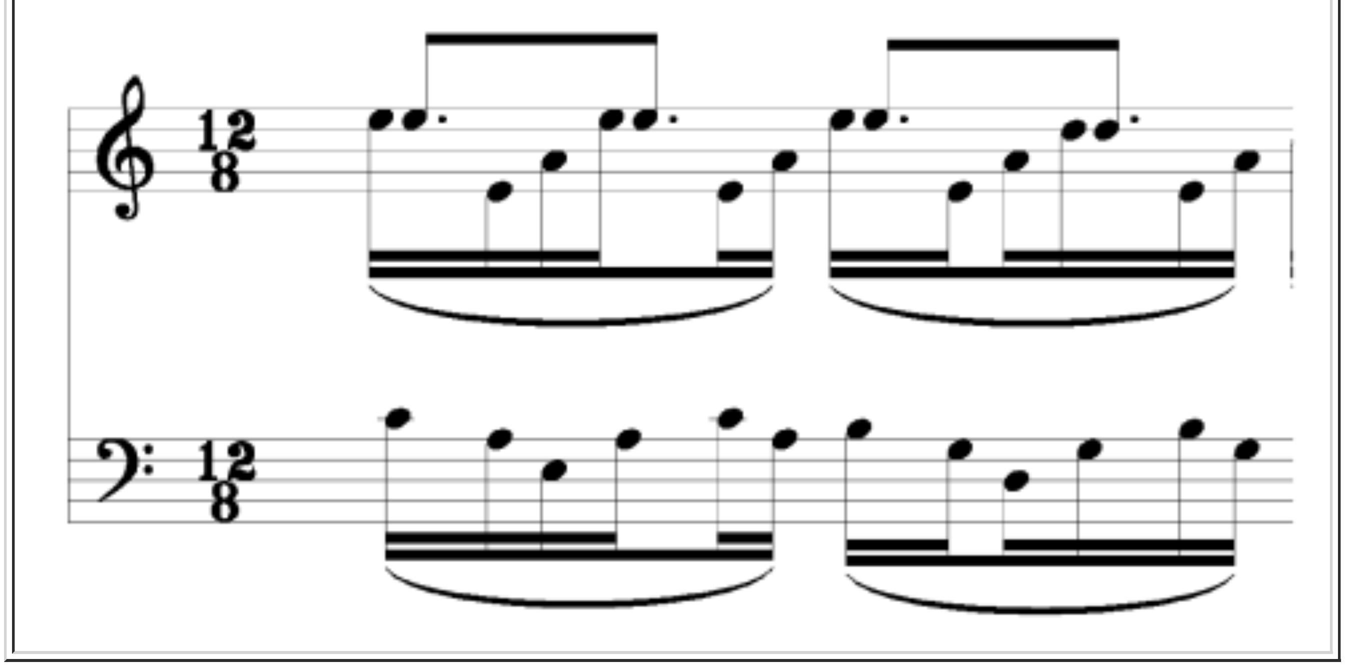
Table 4: Statistics for assembly of primitives in ``Bonita''

The assembly stage has achieved near 100% accuracy because it is very easy to distinguish which grammar production should be applied due to CMN being so structured. The only errors are due to mis-recognised objects from PRIMELA. For example, the single ``note with beam'' error in Table 3 was due to the OCR system leaving a number above a beamed note, which PRIMELA mistook for the top of a long stem that extended above the beam.

Figure 7 shows one of the original pieces of music, and the scanned representation from CANTOR that has been put into a music editor. The system has successfully recognised all of the note-heads, beams, clefs, slurs, and time signature. The features that haven't been recognised are missing primarily because they are yet to be implemented, rather than because the system cannot be adapted to recognise them. The text has also been recognised by the OCR package, but was not passed to the music editor. The stress marks are not present because they are so far away from the note-heads. The upwards stems in the left-hand part are missing because the grammar does not yet have a definition for that construction. It is questionable whether it should be accepted anyway, as it would be represented more accurately by the kind of construction used to show two parts in the top line. Nevertheless, both the stress marks and upwards stems could be added to the grammar if it was considered desirable.



(a)



(b)

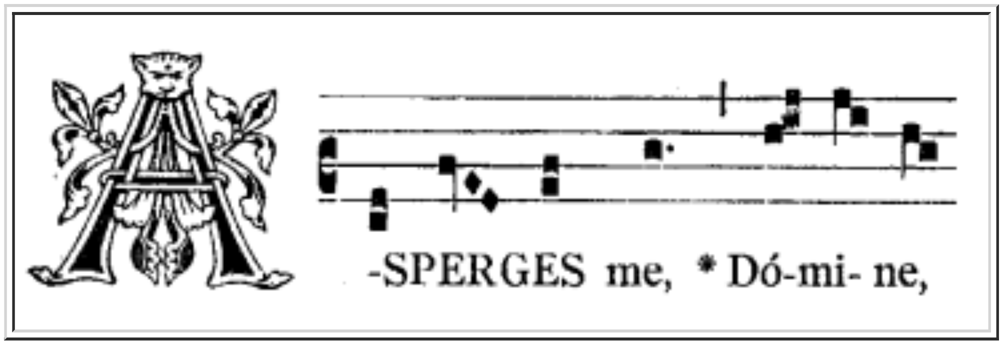
Figure 7: (a) Input to the CANTOR system, and (b) the output displayed in a music editor.

The CANTOR system is currently being extended to include more musical features. A potential problem is that as more features are added, the definition of existing features will have to be refined to ensure that features are distinguished from each other. Our experience so far is that this problem does not become serious.

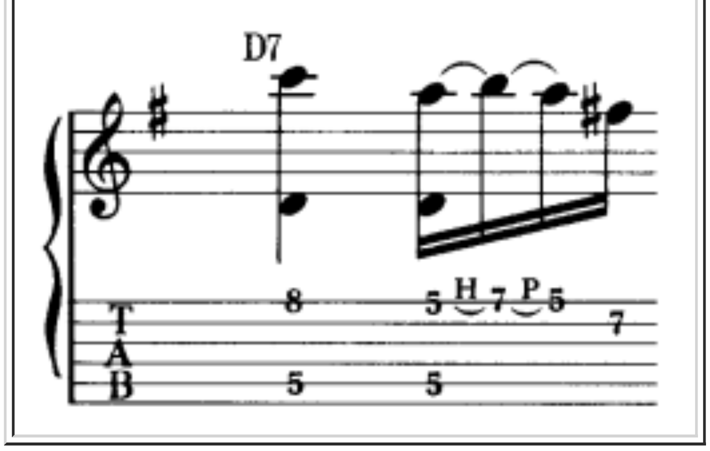
5 Alternative music notations

Most OMR systems are designed to read a subset Common Music Notation (CMN), in which notes are represented using oval heads on five-line staves. CMN is a rich language, and it is difficult to anticipate all of the symbols that might be encountered. CANTOR is easily extended to include new symbols and relationships by adding their description in the PRIMELA language, and defining their function in the DCG.

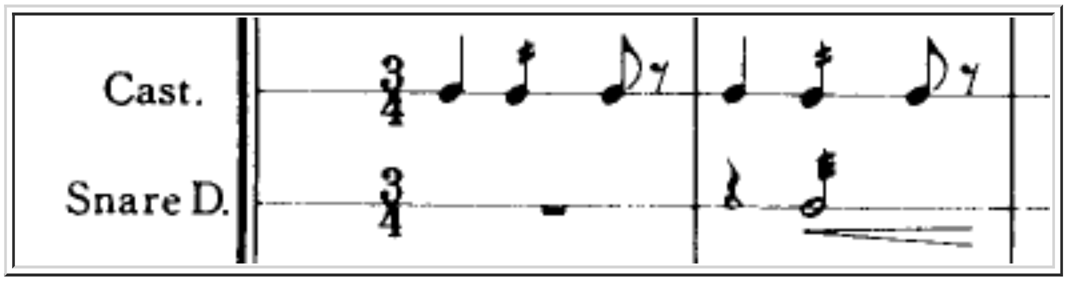
Although CMN is widely used, there are many other types of notation that are used either for different purposes, or in different cultures. As shown in Figure 8, these use varying numbers of stave lines (including 0, 1, 4, and 6), and alternative symbols for note-heads (including squares, crosses, or even numbers or letters).



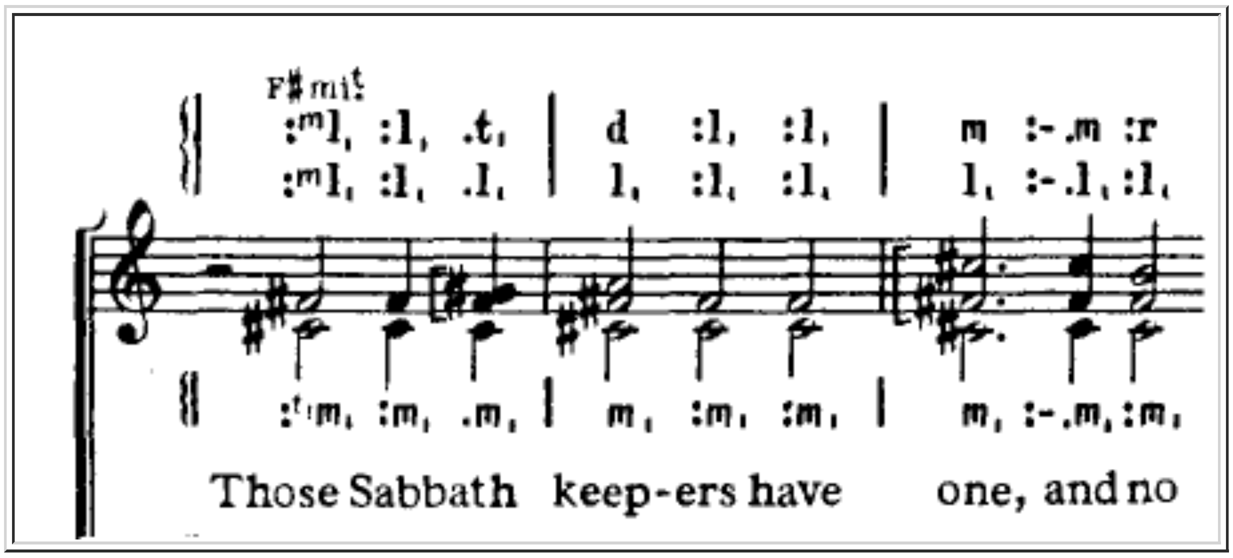
(a)



(b)



(d)



(c)

Figure 8: Some alternative music notations (a) Square-note notation (b) Tablature (c) Percussion (d) Sol-fa.

The CANTOR system was designed to be adaptable so that such alternative forms could also be recognised. The stave removal phase automatically detects the number of lines in the input by searching for groups of regularly spaced lines. Furthermore, it does not assume that the spacing or the number of lines is the same for each group. The PRIMELA language allows new shapes for objects to be defined by the user. For example, square-note notation was accommodated by primarily defining new shapes for note-heads. The DCG was modified to take into account the different constructs, such as the note-heads without stems, and the neumes (grouping of notes that correspond to one syllable of text).

A simple implementation of a square-note specification for music (as in Figure 8a) for CANTOR required 201 lines of PRIMELA code, and 172 lines of DCG code. Generating this code is considerably simpler than modifying a hard-coded recognition program, and allows users to add and modify new features as new pieces of music show up weaknesses. This kind of implementation and modification is only likely to be performed by technically inclined users, but a similar situation works successfully in systems such as Hypercard and the World Wide Web. In these cases, the application is intended for both users and authors, and the distinction is blurred as motivated users are able move towards authoring by making minor alterations to existing work. However, more sophisticated extensions, such as to a new notation or significantly different symbols in an existing one, would need to be carried out by someone with a higher level of technical competence.

In processing an example score that used square head notation, CANTOR correctly identified the 5 clefs and all 121 notes in the work. PRIMELA failed to match one dot, and in another place classified a piece of dirt as a dot; consequently the duration for these notes were incorrect. All pitches were correct. The quality of print was lower than the CMN samples that were tested, so tolerances in the PRIMELA description were more relaxed than for CMN. The processing of uncertain data in the system correctly disambiguated all cases where more than one PRIMELA description matched a particular part of the music.

6 Conclusion

The CANTOR system has demonstrated that an extensible and adaptable OMR system can be implemented by breaking the task into the steps of stave recognition, primitive identification, assembly of primitives, and determining musical semantics. The system currently recognises a subset of CMN reasonably accurately, and our experience is that it is not difficult to extend the size of the subset that can be recognised. The system has also been adapted to read square-note notation, with high accuracy, and relatively little implementation effort.

Acknowledgements

The authors are grateful to Martin Setchel for helpful advice. This research was partially supported by University of Canterbury research grant number 2201768. The first author has carried out the work with the support of a Commonwealth scholarship at the University of Canterbury.

References

1

A. Arkadev and E. Braverman. *Teaching Computers to Recognize Patterns*. Academic Press, 1967.

2

D. Bainbridge. *Preliminary experiments in musical score recognition*. BEng. thesis, Department of Computer Science, University. of Edinburgh, The Kings Buildings, Mayfield Road, Edinburgh, UK, June 1991.

3

H. Baird, H. S. Bunke and K. Yamamoto (editors). *Structured Document Image Analysis*. Berlin Springer-Varlag, 1992.

4

R. Boyle and R. Thomas. *Computer Vision: A First Course*. Artificial Inteligence Texts. Blackwell Scientific Publications, 1988.

5

I. Bratko. *Prolog: Programming for Artificial Intelligence*. Addison-Wesley Publishing Company, 1990.

6

N. P. Carter. *Automatic Recognition of Printed Music in the Context of Electronic Publishing*. PhD. thesis, University of Surrey, February 1989.

7

A. T. Clarke, B. M. Brown and M. P. Thorne. Inexpensive optical character recognition of music notation: A new alternative for publishers. In *Proceedings of the Computers in Music Research Conference*, page 84 ff, Lancaster, UK, April 1988.

8

B. Coüasnon, P. Brisset and I. Stephan. Using logic programming languages for optical music recognition. In *The Third International Conference on the Practical Application of Prolog*, pages 115-134, Paris, France, April 1995.

9

B. Coüasnon and J. Camillerapp. Using grammars to segment and recognize music scores. In *International Association for Pattern Recognition Workshop on Document Analysis Systems*, pages 15-27, Kaiserslautern, Germany, October 1994.

10

J. D. Foley, A. van Dam, S. K. Feiner and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Publishing Company, second edition edition, 1990.

11

I. Fujinaga. Optical music recognition using projections. MSc. thesis, McGill University, Montreal, CA, 1988.

12

I. Fujinaga. An optical music recognition system that learns. In (Ed) Jacek Maitan (editor), *Enabling Technologies for High-Bandwidth Applications*, pages 210-217, SPIE 1785, 1992.

13

E. Selfridge-Field. Optical recognition of music notation: A survey of current work. *Computing in Musicology: An International Directory of Applications*, Volume 9, pages 109-145, 1994.

Footnotes

...CANTOR

CANTerbury Optical music Recognition

David Bainbridge
Fri Sep 5 17:15:30 NZST 1997