Automating the Explanation of Jazz Chord Progressions Using Idiomatic Analysis

Robert Keller,* Alexandra Schofield,* August Toman-Yih,* Zachary Merritt,† and John Elliott**

*Computer Science Department
Harvey Mudd College
301 Platt Boulevard
Claremont, California 91711, USA
keller@cs.hmc.edu
www.cs.hmc.edu/~keller/jazz/improvisor/
aschofield@g.hmc.edu
August.Toman.Yih@gmail.com
†University of Central Florida
3248 Arden Villas Blvd, Apt. 19
Orlando, Florida 32817, USA
zbmerritt@gmail.com
**14a Merchiston Gardens
Edinburgh EH10 5DD, UK
john@dropback.co.uk
www.dropback.co.uk

Abstract: Musicians interested in improvising melodies often benefit from the ability to analyze chord progressions for tonality and from possession of an understanding of a range of idiomatic chord progressions. We describe automated analysis using an approach to representing idioms known as "bricks," which can also be used in analysis of tonality. The brick nomenclature is attributed to Conrad Cork's "Lego Bricks" approach, as extended by John Elliott. We demonstrate a semi-automatic grammatical approach to analysis of chord sequences and sketch its implementation, which is available in the form of free, open-source software called Impro-Visor.

Introduction

In learning to improvise jazz, it is helpful for the improviser to have an understanding of recurring idiomatic chord progressions and the specific tonalities and keys implied by these progressions (Johnson-Laird 2002). Jazz and popular standards often progress through many tonalities, sometimes six or more in the short span of 32 measures. Without an understanding of progression tonality the improvised melody may sound harsh or amateurish. At least one instruction approach to improvisation (Garcia 2006) emphasizes ignoring specific chords in favor of focusing on keys.

Our interest is in providing automation to help the musician learn, understand, and explain progressions as bigger thoughts than just chords or pairs of chords in isolation. It is helpful for the player to recognize and appreciate certain idiomatic chord progressions, as many progressions recur across the space of tunes. Thinking in terms of idioms, rather than entire tunes, offers intellectual economy, because there are far more tunes than there are idiomatic progressions. Another goal of automation is that, in the long run, computer analysis will result in a more robust and consistent foundation on which humans can agree.

Computer Music Journal, 37:4, pp. 54–69, Winter 2014
doi:10.1162/COMJ_a_00201

Related Work

Our approach is related to grammatical approaches to jazz chord sequences that have been considerably explored in recent years. Ulrich (1977) presented algorithms for analyzing major key centers and chords by starting with notes in the chords expressed as the number of semitones from the bass note. His approach was first to establish the set of keys in which each chord could be present. Then, beginning with single-chord segments, his algorithm iteratively combined adjacent segments into larger segments that could be justified by a single key center, until no further combination was possible. He acknowledged that the result may be dependent on the order of combining pairs, but he accepted a left-to-right order, ignoring other possibilities. Once the key segments are established, a different

algorithm derived the function of chords within that key. Our approach uses a more sophisticated parsing algorithm to identify idiomatic chord progressions and then infers keys from the progressions. We do not accept a left-to-right order by default, but instead use cost-based dynamic programming to determine the best explanation for the sequence. Our approach is grammar-based, whereas Ulrich only uses a grammar for parsing chord symbols, not for chord progressions.

Steedman (1984) proposed a grammar for generating blues chord sequences based on chord names rather than notes. He later (1996) simplified the presentation into a context-free grammar (Chomsky 1956). Chemillier (2004) examined Steedman's approach in greater depth and proposed a number of improvements.

Pachet (2000) surveyed earlier grammar-based approaches and described a number of uses of such analyses, as well as some of the attendant issues. He described a rule-based approach for analyzing progressions, which he stated to be an extension of Ulrich's, sketched above. Although our system is not based on Pachet's work, it shares some aspects with his approach. Pachet's ontology includes "global shapes," such as AABA, ABAB, and blues shapes. Global shape analysis is not one of our current objectives, as we are only interested in capturing *bricks*, our name for local idiomatic progressions. These tend to be eight measures long, or shorter. Pachet's ontology lists five idioms: Turn Around 1, Turn Around 2, Two Five, Two Five One, and Resolving Seventh. Our user-extendable dictionary includes these by different names, and the default dictionary offers a much larger set of idioms. Reflecting the influence of Cork (1988, 2008) and Elliott (2009), our base dictionary includes many varieties of *turnarounds*: Plain Old Turnaround (POT), Suspended POT, Ladybird Turnaround, Foggy Turnaround, Whoopee Turnaround, etc. (some named after tunes in which they occur). We include many varieties of cadences: Straight Cadence, Extended Cadence, Long Cadence, Starlight Cadence, etc., and corresponding *approaches* (the part of a cadence prior to resolution), as well as cadences extended with *dropbacks* (which target a supertonic chord) and *overruns* (i.e., a IV chord following resolution). Many of these bricks are hierarchically defined, as are some units of Pachet's ontology.

Both we and Pachet recognize the cyclic character of jazz choruses, where the final chords of a chorus may resolve back to the first chord. Chord substitutions are included in both approaches. We handle substitutions by a preprocessing phase, or in some cases using the brick dictionary, which allows multiple variations on any specific type of brick.

Our method for parsing chord sequences attempts to find the most likely parse of a complete tune into idiomatic units. A significant problem, to which we devoted attention, is the resolution of ambiguities, due to a given subsequence of chords being explainable in multiple ways. Our solution is to use a modified Cocke-Younger-Kasami (CYK) parsing algorithm (cf. Sipser 2005), followed by a path-cost minimization step, both of which are based on dynamic programming. We assume that Pachet's algorithm, being based on Ulrich's, resolves ambiguity by accepting left-to-right order, which we do not.

Our approach also identifies *joins* (i.e., transitions between bricks, discussed further subsequently). To our knowledge, our work is the first to automate the identification of joins.

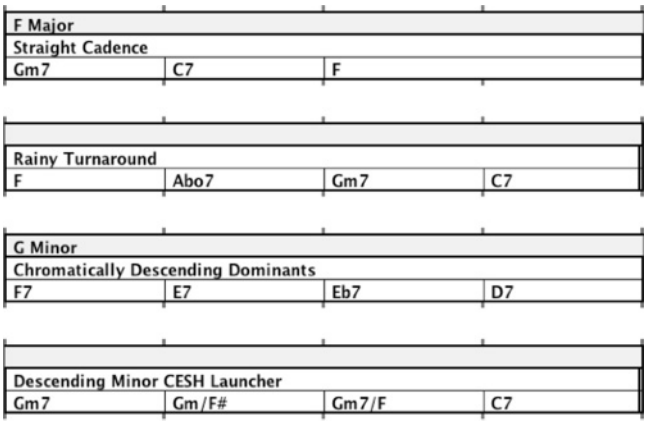
Presentation and ergonomics bring out a final difference between our work and Pachet's. We have tried to construct interfaces for both the display of results and the extension of the idiomatic brick dictionary that would be attractive to the jazz musician not versed in computer science. Our output is a flattened analysis in the form of a "road map," rather than the tree structures used by Pachet. The underlying tree, however, can, in our approach, be explored interactively by decomposing bricks into sub-bricks lower in the hierarchy. We believe that we have been successful on the output side, recognizing that on the input side the dictionary extension aspect could be made more graphically intuitive in future work.

Scholz, Dantas, and Ramalho (2005) were also concerned with a problem similar to ours. They addressed some of the same issues: overlap and ambiguity. They do not elaborate on the parsing approach, but do give a small sampling of rules. Their rules are analogous to ones in our dictionary, but are more complex.

Independently of our use of the CYK algorithm, Wilding (2008) used it in implementing an analyzer for Roman numeral analysis, utilizing Steedman's "categorical grammar" formalism (Steedman 1996), rather than the identification of specific idioms we use.

Choi (2011) provided an algorithm for harmonic analysis by treating tonality segmentation as an optimization problem, deriving lists of Roman numerals for chord functions, with arrows showing approaches and cadences. In our case, these units are further combined into idiomatic bricks that indicate their function.

Figure 1. Four examples of idiomatic bricks. The third track of each row is the input chord sequence, the middle track is the name of the inferred brick, and the top track is the inferred key.



An Approach to Understanding Chord Progressions

A promising approach for understanding jazz progressions was proposed by Cork (1988, 2008) as the "Lego Brick" method. Cork suggested that the chord progressions of many tunes can be structured, and more easily remembered, by dividing the progression into idiomatic bricks. A brick is not a single fixed length, but typically has a duration of one, two, four, or eight measures. For example, a cadence is one example of a brick (although bricks are not limited to cadences). A discussion of the relative merits of the brick representation versus more traditional ones is beyond the scope of this article. Our open-source implementation, however, is intended to be helpful in providing a more rigorous assessment of this approach.

Some bricks can have a major harmonic orientation and others a minor or a dominant one, and these orientations are brought out in the dictionary. Although our dictionary also permits the introduction of relative *temporal space* (e.g., a one-measure versus two-measure II–V component) in brick definitions, our algorithm does not currently attempt to exploit such space. This remains a topic for future refinement of the approach.

Figure 1 presents four examples of idiomatic bricks, as identified by our analyzer. The naming of bricks can be either conventional or colloquial. For example, the name Rainy Turnaround in the second brick of Figure 1 was introduced by Elliott (2009) as common in the tune, *Here's That Rainy Day*. In some cases, the key of a brick is inferred from context rather than the brick in isolation. For example, in the third brick, the chromatically descending dominants could just as well be in G major. However, the progression following being in G minor caused the analyzer to infer the key shown. Table 1 summarizes the chord naming conventions we use. These are based on conventions widely used in jazz.

There are many kinds of bricks, including cadences, launchers, turnarounds, "ons," and more. Many of these terms are widely used beyond the analysis of jazz, or their meaning should be otherwise clear from the context in which they are used. The term *launcher* is perhaps less familiar. It denotes a cadence in which the resolution occurs in a new section separate from, but directly following, the approach. This term was introduced by Cork to suggest that such a cadence "launches" the tune into a new section or phrase. Elliot (2009) provides a more exhaustive discussion of these and other brick types.

Obviously, any brick can be transposed to one of the twelve keys. Cork (1988, 2008) suggests that, in addition to bricks, there are virtual transitions, called *joins*, between bricks. Although some joins can be interpreted as key changes, Cork prefers to describe them as the "magic moment" in the progression, where there is a transition from a stable (i.e., resolved) sound, such as a major sixth or seventh chord, to an unstable sound, such as a minor seventh or dominant seventh chord. The unstable sound may lead to a different stable sound in a different key or tonality, but Cork prefers to think of it as the "sound

Table 1. Chord Naming Conventions

Characters	*Meaning*	*Example Spelling*
b	Flat	Bb = B flat
#	Sharp	F# = F sharp
m	Minor	Cm = {C, E♭, G}
M	Major	C = {C, E, G}
7 (no M or m)	Dominant seventh	C7 = {C, E, G, B♭}
m7	Minor seventh	Cm7 = {C, E♭, G, B♭}
M7	Major seventh	CM7 = {C, E, G, B}
o7	Diminished seventh	Co7 = {C, E♭, G♭, A}
m69	Minor triad with added sixth and ninth.	Cm69 = {C, E♭, G, A, D}
m7b5	Minor seventh with flat 5 (also known as half-diminished)	Cm7b5 = {C, E♭, G♭, B♭}
/	The pitch after the slash is the bass note.	Gm/F# = {F♯, G, B♭, D}

of the transition," rather than the key change, that determines the type of join. In Figure 2, inferred joins are called out with tags below the three tracks. Three joins are shown: New Horizon, Bauble, and Homer. The first of these is common in tunes such as *How High the Moon*, where the tonality steps down a whole step, by way of a major-to-minor transition on the same root. The name "Bauble" is derived from the tune *Baubles, Bangles, and Beads* and is often seen when making a transition from I to III minor. The third join, Homer, is used to cycle back to the same tonality through a II–V progression in the same key, in this case G minor.

There are twelve different joins, corresponding to twelve possible chromatic intervals between pitch classes. These are listed in Table 2.

The work by Elliott (2009) extends Cork's, revealing and classifying a wider variety of bricks that occur in standard tunes. Approximately 21 brick types were identified by Cork; 34 more were identified by Elliott. Furthermore, Elliott's contribution also provided further organization and taxonomic clarification. Elliott gives examples in the form of 241 road maps that represent chord progressions of tunes using bricks and joins. We have adopted Elliott's road map terminology in our software.

Problem Statement

The techniques we present are aimed at solving the following two problems by automated analyses:

1. Given a chord progression, determine a mapping from the chord instances in the progression into keys and tonalities.
2. Given a chord progression, analyze the progression as a series of harmonic idioms called bricks and, where relevant, joins between bricks.

The approach used in our method is to first solve the second problem, then use information about the implied key of the chord progression to solve the first. This is the opposite of Ulrich's (1977) approach, which tries to establish the key first and then identify the function of chords within the key.

The repertoire of possible bricks will be predefined in a table called the *brick dictionary*, as it provides an association between the name of the brick and the components of the brick. Because a given name can have more than one realization as a chord sequence, this dictionary functions as a grammar (Chomsky 1956). Components of bricks can be either chords or other bricks. By repeated expansion, any brick can eventually be translated to a sequence of chords.

Analysis Example

Our first example is a standard tune, *Bye Bye Blackbird*. The chord progression is shown in Figure 3. The input as shown is "lead sheet" notation (Keller 2005), designed for the convenience of jazz musicians. The vertical bars in the table signify divisions between

Figure 2. Three examples of joins, shown as tags below the bricks (New Horizon, Bauble, and Homer), as inferred by our analyzer from typical chord sequences.

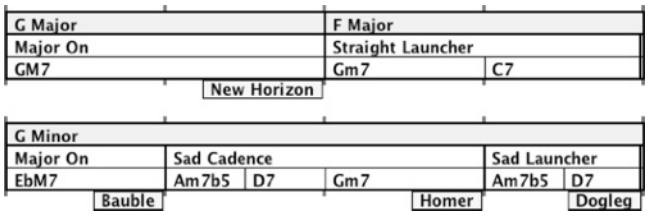


Table 2. List of Joins

Semitones between I and the New IIm7 Chord	*Join Name*
0	New Horizon
1	Downwinder
2	Homer
3	Cherokee
4	Woody
5	Highjump
6	Bauble
7	Bootstrap
8	Stella
9	Backslider
10	Half Nelson
11	Sidewinder

The join names Cherokee, Woody, Bauble, Stella, and Half Nelson are taken from the names of jazz tunes; the other names are arbitrary.

measures. A slash by itself indicates repetition of the previous chord. The chord naming conventions used in this article were given in Table 1.

The output of the analysis is shown in Figure 4. The lead sheet in Figure 3 includes section markers, which are helpful and sometimes necessary to reduce ambiguity in the analysis. In the present example, however, the additional help is very minor.

As can be seen in the road map in Figure 4, both tonality information (in the upper track) and idiomatic brick information (in the middle track) are obtained from the lead sheet input (the bottom track). A brick name ending with "+ . . ." (as in the first brick of line 2) means that the last chord of the brick is shared with the first chord of the next brick. In this example, the Gm7/F is shared between the bricks labeled "Descending Minor CESH" and "Two Goes Straight Cadence." We call this shared chord a *pivot chord*.

The analyzer correctly identifies the sequence Gm7 | Gm/F♯ | Gm7/F in lines 2 and 3 as a Contrapuntal Elaboration of Static Harmony (CESH) brick, using the terminology of Coker (1997). As previously stated, POT stands for Plain Old Turnaround, a term used by Aebersold (1979), and in Cork's terminology SPOT stands for Suspended POT, meaning that a III chord occurs where the initial I chord would occur in a POT.

Brick definitions are often based on empirical observations of common idioms used by composers of jazz and popular tunes. In our analysis, bricks are defined in a dictionary, and the end result of the analysis is strongly dependent on the dictionary content. We use as our baseline the set of bricks defined in Elliott (2009), with a few modifications and additions.

The brick dictionary can be viewed as a grammar (Chomsky 1956) in the following sense. The terminal symbols of the grammar are the chord symbols. The non-terminal symbols of the grammar are the brick names, accompanied by certain qualifying information, such as the implied key or tonality. Each dictionary entry is effectively a production rule in the grammar. Figure 5 shows a few examples of rules from the dictionary in textual form. Following the name of the brick, the tonality (mode, i.e., major or minor), brick category (definable by the user), and relative key, is a sequence of items, each of which is either a chord or name of another brick. Both specify the relative duration (although that information is not currently exploited) and, in the case of a brick, must specify its key. Only one rule is included for all twelve possible tonics. The parsing algorithm handles the transposition of the rule to the relevant key of interest.

A dictionary capable of analyzing most tunes in the standard jazz repertoire can be constructed from about 500 such definitions. A form of grammatical non-determinism is provided by certain bricks having multiple definitions, each distinct definition accompanied by a qualifier to remove ambiguity in an explanation.

Analysis Technique

We have automated the analysis of chord bricks to provide an in-depth analysis of the structure of

Figure 3. Lead sheet notation for Bye Bye Blackbird.

Figure 4. Analysis of the chord sequence from Bye Bye Blackbird, *with input from lead sheet notation.*

```
(section) FM7 | Gm7 C7 | F6 | / | F6 | Abo7 | Gm7 | C7 |

(section) Gm7 | Gm/F# | Gm7/F | C7 | Gm7 | C7 | F6 | / |

(section) F7 | E7 | Eb7 | D7 | Gm7 Gm/F# | Gm7/F Gm/A | Gm7 | C7 |

(section) FM6 | Gm7 C7 | F6 | Am7b5 D7 | Gm7 | C7 | F6 | Gm7 C7 |
```

Figure 3

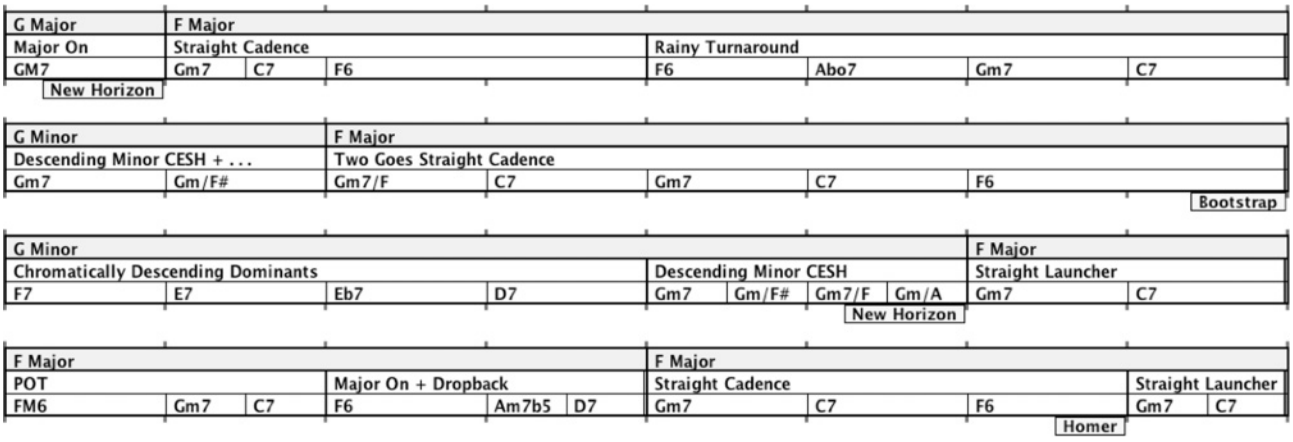


Figure 4

chord progression in a jazz tune. In our model, a brick can be a sequence of just chords, or it can be a sequence of chords and other bricks. This generality provides the possibility of hierarchical classification, in which complex bricks can be understood in terms of simpler ones. A single chord differs from a brick in that it has not been given a function, whereas a brick provides a function, such as a cadence, in addition to the constituent chords. Thus a brick is generally represented as a sequence of items that may be just chords, smaller bricks, or both. (Elliott [2009] used the term "meta-brick" to denote a brick composed of other bricks, but we have not used this term in the current presentation.)

Figure 6 illustrates how a brick can be hierarchically decomposed. The starting brick is the opening to *Autumn Leaves*. It decomposes into two cadences, which can be further decomposed into their respective approach and resolution blocks. Continued decomposition, until none is further possible, is referred to as "flattening" the brick. The analysis essentially reverses the decomposition process, starting with chords and composing them into increasingly coarser bricks. A major issue is that more than one such combination is often possible. We need to find the one combination that best explains the tune.

Our analysis is done in two stages. The first stage parses chord progressions into bricks and returns the perceived best brick sequence for the progression. The second stage uses the brick sequence to determine the key map and joins. It also may modify some of the bricks using knowledge of keys.

Grammar Rules

The grammar of bricks is based upon the idea of having chords as terminal symbols and bricks as non-terminal symbols. The Cocke–Younger–Kasami (CYK) parsing algorithm (cf. Sipser 2005) appears to be a good means of parsing bricks due

Figure 5. Sample brick representations in the dictionary. Each definition is an S expression that gives the name, optional variant (in parentheses), mode, brick type, resulting key, and list of constituent blocks.

```
(defbrick Perfect-Cadence Major Cadence C
          (chord G7 1)
          (chord C 1))
(defbrick Straight-Approach Major Approach C
          (chord Dm7 1)
          (chord G7 1))
(defbrick Straight-Cadence Major Cadence C
          (brick Straight-Approach C 2)
          (chord C 2))
(defbrick POT Major Turnaround C
          (chord C 1)
          (chord Am7 1)
          (brick Straight-Approach C 2))
(defbrick Extended-Approach Major Approach C
          (chord Am7 C 1)
          (brick Straight-Approach C 2))
(defbrick Extended-Cadence Major Cadence C
          (brick Extended-Approach C 3)
          (chord C 1))
(defbrick Minor-On(main) Minor On C
          (chord Cm 1))
```

Figure 5

Figure 6. Example of a block hierarchy with successive decomposition of bricks. The first line shows an eight-chord brick, the second line shows the decomposition into two sub-bricks that are cadences, the third line shows further decomposition of those bricks, and the last line shows all bricks flattened out to the chord sequence.

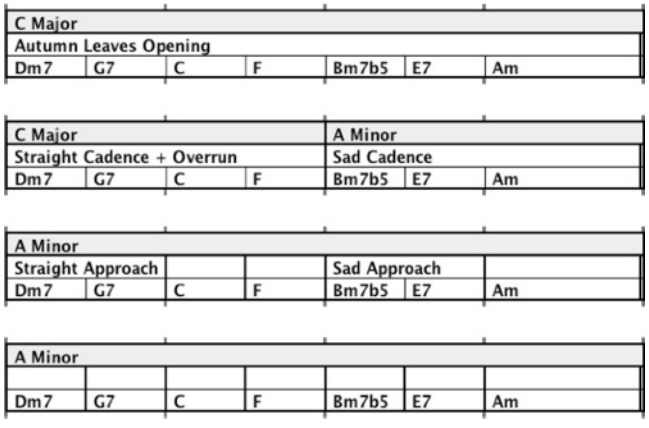


Figure 6

to its flexibility in parsing ambiguous grammars. The standard precondition of applying the CYK algorithm, however, is that the grammar must be expressed in Chomsky normal form (CNF), in which every production rule must be one of two types:

1. Binary: A → B C, a non-terminal producing two non-terminals
2. Unary: A → α, a non-terminal producing one terminal symbol

For convenience in handling substitutions of chords and progressions in jazz sequences, we permit a third production type not included in CNF:

3. Unary: A → B, a single non-terminal producing a different non-terminal

It is easy to see how to extend the CYK algorithm to accommodate the additional third case.

Rules in the user's brick dictionary may have more than two items on the right-hand side, but it is a standard exercise to preprocess such rules by introducing additional non-terminals as necessary (cf. Sipser 2005).

Parsing Algorithm

The standard CYK algorithm determines whether or not there is a parse of an input string from a single designated start symbol. Our objective is slightly different. We do not attempt to have a single start symbol that will generate every tune. Instead, our objective is to parse a sequence of non-terminals, each standing for a brick or a single chord. The process is described as constructing a tree. The CYK algorithm produces a forest (set of trees), and the cost minimization phase combines trees selectively to produce a single tree. Trees are constructed of nodes that have zero, one, or two children.

In the initial stage of parsing, the i^{th} chord terminal is converted into a node representing the chord and placed on the diagonal entry (i, i) of the parse table. If there are substitutions for the chord, nodes for these are added to the diagonal entries. The rationale for these substitution nodes is that, to avoid a combinatorial explosion of the dictionary size, we do not want to maintain dictionary rules for every possible substitution. Instead, we rely on representative chords that accommodate the set of possible substitutions. For example, although a rule in the dictionary might include a G7 chord on the right-hand side of a rule, many other chords could be substituted for the G7, including G9, G13, G7♭9, G7♯9, G7♭9♯11, etc. Providing separate lists of

substitutions for common chord types prevents this explosion. Another way of handling it would be to use a single non-terminal standing for all members of a family of substitutions. We think it is more intuitive for the musician using the system to work with a representative member than another abstract symbol.

The development of a parsed sequence of bricks from a series of chords is then done in the following steps:

1. For an input chord sequence $C_1\ C_2\ \ldots C_n$ of length n, an $n \times n$ table T is constructed. Each diagonal entry (i, i) of T is initialized with a single-node tree representing chord C_i.
2. The parser applies the CYK algorithm to fill the table entries above the diagonal (i.e., those entries $[i, j]$ where $i < j$) with trees. Each tree represents a parse of all contiguous subsequences of chords $C_i \ldots C_j$. Each child of a given node is a root of one of the sub-trees that combine to make that node's tree. The algorithm iteratively computes all entries $T(i, i+1)$ for each i, then $T(i, i+2)$, etc., until $T(1, n)$ is computed. The computation of a table entry $T(i, j)$, for $i < j$ is computed by systematically iterating through pairs of entries $T(i, k)\ T(k+1, j)$ for each k, where $i \leq k \leq j$. For each such pair, the parser examines all the production rules of Type 1 (from the classification of production rules in the previous section) to see if the pair of non-terminals in the body of the production corresponds to those in the currently selected pair. Extra considerations are required beyond the standard CYK algorithm to connect sequences on the right-hand side by the correct "relative key difference," and to derive the key attribute of the left-hand side non-terminal. (Because the grammar rules are each given in one key, they need to be transposable to any of the twelve keys. The relative transposition is determined when the first element of the right-hand side is matched against a node in the table. For example, a rule for an approach brick Dm7 G7 in the key of C will match the chord sequence Fm7 B♭7 in the key of E♭ by transposing Dm7 three semitones to Fm7. Then G7 is transposed by the same three semitones to match the B♭7.)
3. A path-finding algorithm determines a minimal-cost combination of the parses in the table that describe the whole of the tune.
4. Additional processing touches up the result in a manner to be described subsequently.

A simple example is worked out in greater detail in Appendix 1.

Handling Rule Overlap

An additional complicating factor of many chord sequences is that some patterns may overlap with each other. For example, the end chord of one brick may act as a pivot chord, starting another brick. We saw examples of this in the second line of *Bye Bye Blackbird* mapped in Figure 4. To account for such overlap possibilities, whenever a new non-terminal is placed into $T(i, j)$, a nearly identical non-terminal is also placed in $T(i, j-1)$. This second non-terminal has, however, a final chord with zero duration, accounting for the presence of the chord in the brick without changing the duration of the overall piece.

Finding a Minimal-Cost Explanation from the Parse Table

In order to determine the quality of a given parse based upon the bricks identified in the preceding step, the parser assigns a cost to each brick type, ultimately favoring the sequence with the lowest overall cost. Individual brick costs are determined heuristically by a few factors. The base cost comes from the type Cadence, which is assigned a cost of 25. The cost is then adjusted slightly in the case of an overlap or a substituted chord. Table 3 shows the costs we currently use, although the user of our analysis tool is free to specify a different table.

The costs in Table 3 were determined starting with a general idea of preferred types of bricks,

Table 3. Typical Cost Assignments for Selected Types of Bricks

Brick	*Cost*
Approach	45
Cadence	25
CESH	10
Deceptive Cadence	60
Dropback	30
Ending	30
Invisible	2,000
Misc	40
Off	1,000
Off-On	1,010
On	550
On-Off	1,005
On-Off+	100
Opening	25
Overrun	30
Pullback	45
Turnaround	20

then modified by designer trial-and-error. We present some of the rationale here. The more generalized bricks and most singleton chords are placed two orders of magnitude away, by cost, from the preferred bricks for analysis. For example, On bricks, which represent tonic single-chord bricks, cost much less than non-tonic Off single-chord bricks, which are less stable.

A turnaround brick is made somewhat less costly than a cadence due to a tendency for a series of turnarounds to parse otherwise as a series of cadences, a slightly less informative parse that would possess the same number of bricks. The cost of an approach (the first part of a possibly incomplete cadence, such as a II–V approach in a II–V–I cadence) is made somewhat higher due to a preference for bricks that resolve in the form of a cadence when possible. The cost-minimization algorithm is summarized in Figure 7. It is similar to Floyd's (1962) algorithm for finding minimum cost paths in an edge-labeled graph.

Due to the order in which the algorithm updates its mins table, entries in the updating step will never be empty. After cost minimization, the top-right cell of the table of minimum-cost tree nodes will have a top-level tree node containing the lowest-cost explanation of the piece. Such a node will always exist. In the worst case there will always be a finite-cost solution for the parse in the form of the original chords. This tree is then reassembled into a sequence of blocks as the basis for the road map.

Post-Processing and Key Analysis

After the parser returns a sequence of blocks, the results are refined via a series of post-processing steps that find spanning keys of sections in the tune, as well as launchers and joins.

Although each brick (counting isolated chords as one-chord bricks) has its key stored at the end of the parse, most tunes stay in the same key for more than one consecutive brick, necessitating an accurate way to track the overall key progression of the tune. To accomplish this tracking, the key, mode, and duration of each portion of the tune determine a series of "key spans." Currently we use only three generalized modes: major, minor, and dominant (also known as mixolydian), sacrificing some specificity in favor of simplicity.

To determine the length and characteristics of a key span, the list of bricks in a tune is traversed from the end of the tune backward to the start, comparing the key and mode of the current brick to the tonic of the key span immediately following it. If the brick in question can be merged into the key span following, merging is done and traversal continues. If not, a new key span is initialized using the current brick's key, mode, and duration. Traversal continues backward until the beginning of the tune is reached.

The method of key comparison differs slightly depending on whether we have a "true" brick or an isolated chord treated as a brick. For a true brick, a simple check occurs to see if the key and mode of the brick equal those of the current key span. If so, the brick is merged into the current key span. A special case is also considered in which the brick is an approach that resolves to the key span. This was added to deal with *surprise cadences*, the term used by Cork (2008) to designate cadences that resolve to a different mode than that of the approach leading up to it.

Figure 7. Algorithm for finding the minimum cost parse. The variable cykTable *is the array of all possible nodes in lists after the CYK algorithm runs.*

```
mins = new Array<Node>[cykTable.length][cykTable.length]

for row = 0 to mins.length - 1:

    for col = row to mins.length - 1:

        mins[row][col] = the lowest cost Node in cykTable[row][col]

for i = mins.length - 1 to 0 by -1:

    for j = i + 1 to mins.length - 1:

        for k = i + 1 to j:

            if mins[i][k-1].cost + mins[k][j].cost < mins[i][j].cost:

               mins[i][j] = new Node(mins[i][k-1], mins[k][j])

            end if
```

Analyzing a solitary chord for possible merging into a key span requires additional infrastructure, because one chord could be in a number of different keys, depending on its scale degree. Each chord is checked against a list of chords commonly used in a given key and mode. These lists are maintained for minor, major and dominant modes in the key of C and transposed to arbitrary keys as appropriate.

Launchers and Joins

To locate launchers, each block ending a section or phrase is compared with the one beginning the section immediately following, to test whether the first block is an approach that would resolve to the second block. The last block in the tune is compared with the first one, because choruses in jazz tunes are typically repeated to allow for extended improvisation, with the end of the chorus usually making a transition back into the beginning.

A Straight Launcher is simply a Straight Cadence split over a harmonic phrase or section. For comparison, Figure 8 illustrates a cadence versus a launcher. Notice the double bar in the middle of the bottom half, indicating a phrase break. In this case, there are two bricks rather than one, as in the top half.

The process of finding launchers and joins is done by iteration through all of the blocks. A second purpose is also served: finding launchers that resolve "specially." The approach component of a brick such as a Yardbird Cadence (Fm7 B♭7 C), when examined without its resolution, is the same as a Straight Approach in a different key, E-flat. In post processing, each block is compared to the next one to identify any such specially resolving approaches and convert them to launchers. Figure 9 shows an example of a Yardbird Launcher. (The terminology is derived by Cork from Charlie Parker's tune *Yardbird Suite*, although it is also known as a "back door" cadence and as a minor plagal cadence.)

Comparison with Manual Annotation

The application of Cork's approach to analyzing tunes is still relatively unknown in jazz circles. The original work analyzed about twenty jazz tunes. The most comprehensive study to date is that by Elliott (2009), which analyzed 241 tunes. Elliott extended the set of bricks significantly, and made a strong effort to clarify Cork's approach in an accompanying book. All analyses prior to the present article were

Figure 8. Cadence versus launcher. The double bar in the bottom line indicates a section or phrase break, which a brick is not allowed to span. Therefore the cadence is divided into a launcher and an "On" brick representing the resolution of the progression.

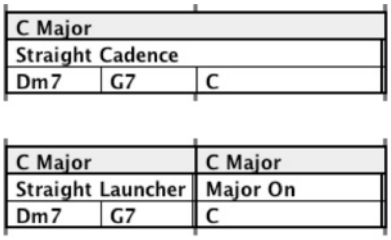
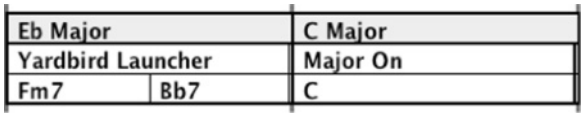


Figure 9. Special resolution for a Yardbird launcher. The expected resolution of the launcher is E♭, but because it resolves to C in this case, it is called Yardbird, as exemplified by the tune Yardbird Suite.



done manually, by and for humans with musical experience.

It is worthwhile to clarify the roles of the co-authors at this point. The first four authors designed and implemented the software system. The last author, John Elliott, was not involved in the development directly, but provided occasional critique and clarification, as we were improving our understanding of the meaning of the bricks as the research progressed. We tried to adopt most of Elliott's brick definitions as a basis for our dictionary. In some cases, however, we made extensions of our own, and in others we modified definitions in an attempt to make the analyses consistent. Computer analysis is more demanding than manual analysis, and makes it more difficult to introduce informal characterizations "on the fly."

We now describe the activity of comparison. We used the first 103 of Elliott's 241 manual analyses as a base (cf. Elliott 2009). We used the analyses labeled B1 through B100. Analysis B25 was four distinct blues variations, making the total 103. As these were published in alphabetic order, we assumed there would be no bias from simply choosing the first hundred. We should point out that the chord progressions were not necessarily the same ones found in standard "real books." Rather, a certain amount of personalization had been done on some of the tunes, which in retrospect had a complicating effect on the analyses. Section markers were included in the tunes. For example, a typical AABA tune would have four eight-bar sections. The main effect of section markers in the analysis is that bricks are not allowed to straddle sections, thereby removing some opportunities for ambiguity in harmonic phrasing. In a few cases, following the section markers in Elliott's road maps, four-bar sections rather than eight-bar sections were indicated.

A first pass was made using Impro-Visor to analyze the same changes and produce road maps according to our method. After viewing the results about one third of the tunes were redone. Additional phrase marks were added in 25 tunes and phrase marks were removed in 3 tunes to correct harmonic phrasing. The justification for these actions is that, although a lot of phrasing is de facto inferred, Impro-Visor is not generally capable of inferring harmonic phrasing based just on the chord sequence. In addition, some bricks that were obviously missing from the dictionary were added. One other hint that was given in two cases was to indicate that a dominant chord should be interpreted as a tonic, another facet that cannot generally be inferred. Impro-Visor uses the convention that a dominant chord symbol with an underscore appended should be interpreted as a tonic.

Having made these corrections, Keller and Elliott each privately evaluated Impro-Visor's performance on the 103 tunes, comparing them with the manually produced road maps as a previously published "standard." Tunes for which both analyses were essentially the same, or equally valid, received a rating of zero. Tunes where the Impro-Visor analysis differed from the standard received negative ratings, one point off for each incorrect analysis component, but not counting multiple recurrent errors of the same type. The number of features for comparison ranged from 4 up to 20 or more. The range of negative ratings was −1 to −3 by Elliott and −1 to −4 by Keller. In three cases, Elliott evaluated the Impro-Visor road map as better than the manual ones, and those received a +1 rating.

The average of the manual analyzer's (Elliott's) scores was −1.04 with a standard deviation of 0.9, and the average of Keller's scores was −1.17 with a standard deviation of 0.95. Table 4 shows the number of tunes with each rating, as rated by Elliott. Keller's ratings were similar, but slightly lower in a

Table 4. Rating of Impro-Visor's Automatic Performance versus Manual Annotation

Rating	*Number of Tunes*	*Percent of Total*
+1	3	3
0	27	26
−1	39	38
−2	31	30
−3	3	3

few cases. The second appendix gives the complete list of tunes by rating.

Detailed comparison indicates some ways in which the automatic analysis can be improved. A major missed opportunity seems to be in not taking hints from harmonic tempo (also known as harmonic rhythm). In some cases there is one chord change every measure and in others every two measures. Often the boundary between two bricks is signaled by a change in the harmonic tempo, with most chords within the brick staying in the same tempo. Being able to infer such harmonic tempo changes could then be exploited as a sufficient condition for a transition to a new brick. It would not be a necessary condition, however, so there would still be cases of harmonic phrasing that cannot be inferred by this method.

Another possible improvement would be to increase the sophistication of key recognition. Because key recognition is currently driven by inference from bricks after they have been identified, it is possible that even a somewhat lengthy chord sequence could identify the wrong key. A glaring example occurs in *Alice in Wonderland*, where Impro-Visor identified the chord sequence F♯m7♭5 B7♭9 Em7 A7 as a SPOT in D major, whereas the entire rest of the piece was C major. The SPOT should have been seen as the first part of a Starlight progression, which would continue with Dm7 G7, establishing the key as C major, consistent with the rest of the tune. The concept of a brick being truncated part way through occurs in several of Elliott's analyses, but Impro-Visor has no provision for truncated bricks.

Figures 10 and 11 show a relatively successful comparison of our algorithmic analysis with Elliott's

Figure 10. Elliott's (2009) analysis of Blue Bossa. *Although slightly different chord symbols are used, their meaning is the same as the symbols in Figure 11.*

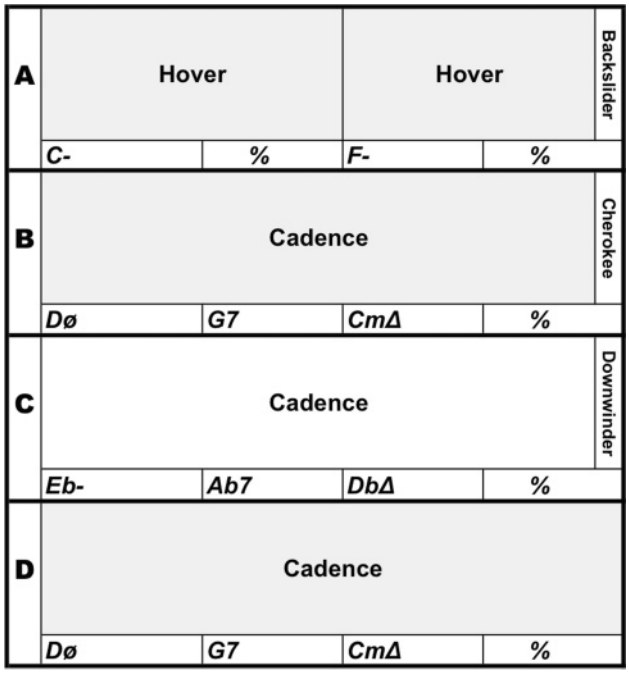


Figure 10

Figure 11. Impro-Visor's analysis of Blue Bossa.

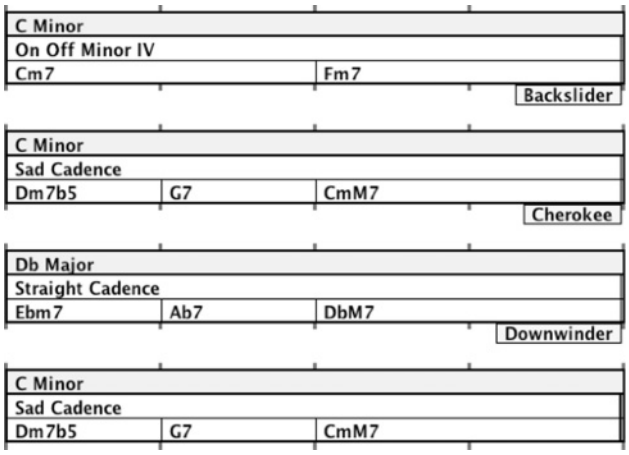
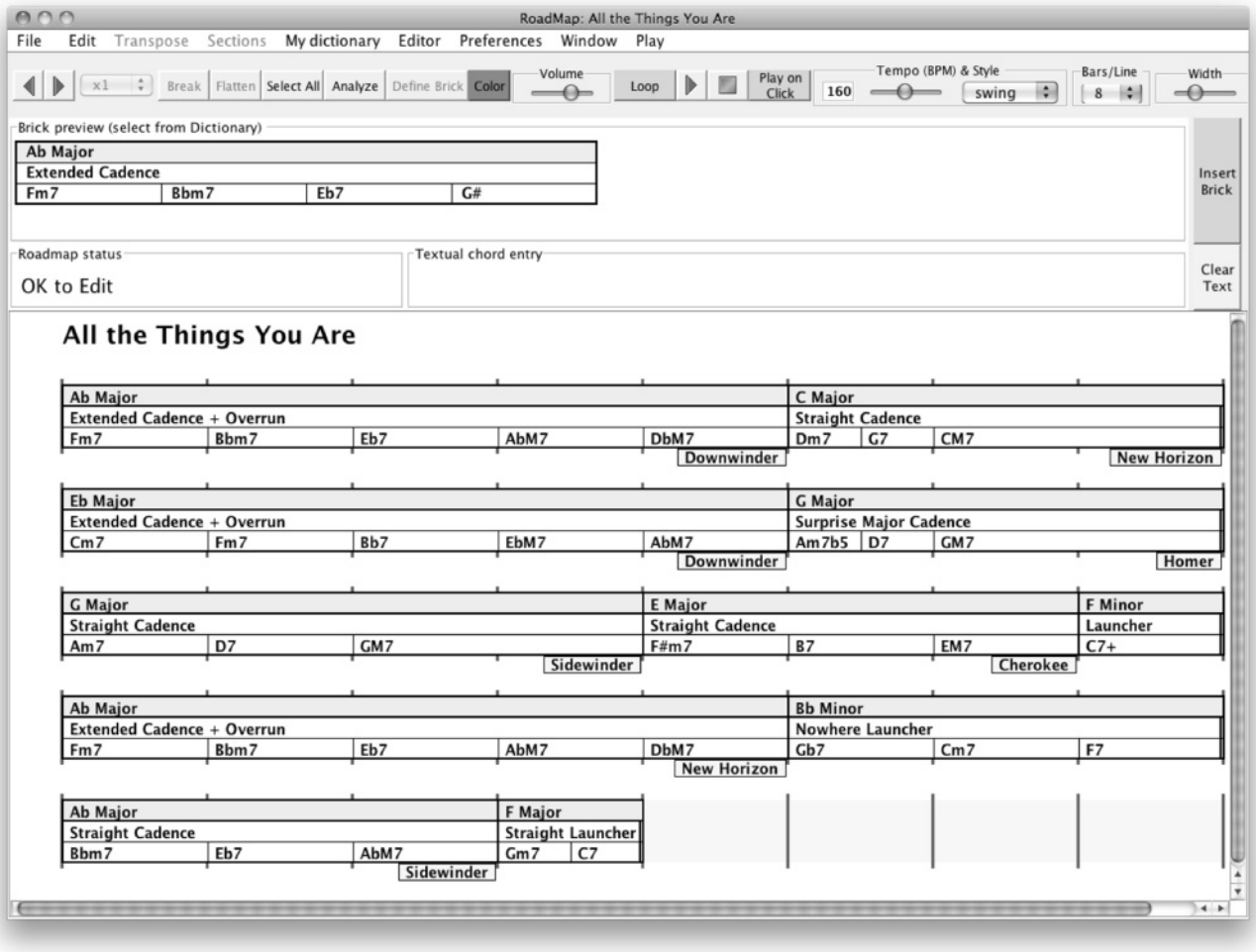


Figure 11

manual analysis for the tune *Blue Bossa*. The analyses are essentially the same, as all the joins and cadences are identified the same in both. Our vocabulary does not include the concept of "hover," which means to dwell on the same chord for a significant time, so in our analysis, the first line appears as a single On-Off brick rather than two

Figure 12. User interface of the interactive road map displayed by Impro-Visor.



separate bricks. Furthermore, Elliott has indicated that he agrees with Impro-Visor's analysis.

Conclusion

We have presented an analysis technique for creating an explanation of a tune's chord sequence using idiomatic bricks and keys. The key analysis begins by first finding bricks, which establishes their keys based on a brick dictionary. Issues of ambiguity and overlap are addressed by assigning relative costs to brick types. We have shown how our technique can determine the bricks and joins in a tune, as suggested by Cork. We use key information defined along with bricks to determine the sequences of keys and tonalities for the tune. All information is presented in a structure called a road map. Such information is deemed useful for improvisers and accompanying musicians. Space limitations did not permit detailed description of our interactive tool for computing, displaying, and manipulating road maps, but we provide a screen shot of the tool in Figure 12. The tune being analyzed, *All the Things You Are*, is seen to have six distinct keys. It also illustrates a variety of brick types and four types of joins, helpful in understanding how the tune is knit together. This tool is available as the free

open-source software Impro-Visor (Keller et al. 2012). In addition to providing a representation of the analysis, including the ability to break large bricks into constituent parts, it also permits interactive playback and construction of new progressions from the brick dictionary.

Acknowledgment

This work was supported in part by the National Science Foundation through grant CNS REU #0753306.

References

Aebersold, J. 1979. *Turnarounds, Cycles, and II/V7's*. New Albany, Indiana: Jamey Aebersold Jazz.

Chemillier, M. 2004. "Toward a Formal Study of Jazz Chord Sequences Generated by Steedman's Grammar." *Soft Computing* 8(9):617–622.

Choi, A. 2011. "Jazz Harmonic Analysis as Optimal Tonality Segmentation." *Computer Music Journal* 35(2):49–66.

Chomsky, N. 1956. "Three Models for the Description of Language." *IRE Transactions on Information Theory* 2(3):113–124.

Coker, J. 1997. *Elements of the Jazz Language for the Developing Improviser*. Los Angeles, California: Alfred.

Cork, C. 1988. *Harmony by LEGO Bricks*: *A New Approach to the Use of Harmony in Jazz Improvisation*. Leicester, United Kingdom: Tadley Ewing Publications.

Cork, C. 2008. *The New Guide to Harmony with LEGO Bricks.* London: Tadley Ewing Publications.

Elliott, J. 2009. *Insights in Jazz*: *An Inside View of Jazz Standard Chord Progressions.* Available online at dropback.co.uk. Last accessed 20 May 2013.

Floyd, R.W. 1962. "Algorithm 97: Shortest Path." *Communications of the ACM* 5(6):345.

Garcia, A. 2006. *Cutting the Changes*. San Diego, California: Kjos Music.

Johnson-Laird, P. N. 2002. "How Jazz Musicians Improvise." *Music Perception* 19(3):415–442.

Keller, R. 2005. "Leadsheet Notation." Available online at www.cs.hmc.edu/~keller/jazz/improvisor/LeadsheetNotation.pdf. Last accessed 20 May 2013.

Keller, R., et al. 2012. Impro-Visor. Available online at sourceforge.net/projects/impro-visor/. Last accessed 20 May 2013.

Pachet, F. 2000. "Computer Analysis of Jazz Chord Sequences: Is Solar a Blues?" In E. Miranda, ed. *Readings in Music and Artificial Intelligence*. Amsterdam: Harwood Academic Publishers, pp. 85–113.

Scholz, R., V. Dantas, and G. Ramalho. 2005. "Automating Functional Harmonic Analysis: The Funchal System." In *Proceedings of the Seventh IEEE International Symposium on Multimedia*, pp. 759–764.

Sipser, M. 2005. *Introduction to the Theory of Computation.* 2nd ed. Independence, Kentucky: Cengage.

Steedman, M. 1984. "A Generative Grammar for Jazz Chord Sequences." *Music Perception* 2:52–77.

Steedman, M. 1996. "The Blues and the Abstract Truth: Music and Mental Models." In A. Garnham and J. Oakhill, eds. *Mental Models in Cognitive Science*. Mahwah, New Jersey: Erlbaum, pp. 305–318.

Ulrich, W. 1977. "The Analysis and Synthesis of Jazz by Computer." In *International Joint Conference on Artificial Intelligence*, pp. 865–872.

Wilding, M. 2008. "*Automatic Harmonic Analysis of Jazz Chord Progressions Using a Musical Categorical Grammar.* " M.Sc. thesis, University of Edinburgh, School of Informatics.

Appendix 1: Parse Table Example

In Table 5 we present an excerpt of a parse table produced by our algorithm. Most of the grammar rules used here were defined in Figure 5, except for partial POT and dropback. The former is a generated rule corresponding to breaking down the POT production into simpler parts to achieve Chomsky normal form. The dropback is built into the system, rather than being defined explicitly, as every cadence potentially can be extended to one with an added dropback (VI7 or VIm7) and there are many of types of cadences. Table 5 underlies the production of the road map, but is not meant for general user consumption. The table covers the first four measures of the tune *Blue Moon*. The chord sequences in square brackets serve to indicate the chord sub-sequences represented by the table entries. The diagonal entries show the original chords and non-terminals derived from them by unary productions. Above the diagonal are other derived non-terminals, derived

Table 5. Example of a Parse Table

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
0	[C6] C Major On	[C6 Am7] C On + Dropback C Major On Off VI A Minor Off On bIII	[C6 Am7 Dm7] C partial POT	[C6 Am7 Dm7 G7] C POT	[C6 Am7 Dm7 G7 C6] C POT + On	[C6 Am7 Dm7 G7 C6 Am7]	[C6 Am7 Dm7 G7 C6 Am7 Dm7]	[C6 Am7 Dm7 G7 C6 Am7 Dm7 G7]
1		[Am7] A m7 A Minor On	[Am7 Dm7] A Minor On Off IV	[Am7 Dm7 G7] C Extended Approach	[Am7 Dm7 G7 C6] C Extended Cadence	[Am7 Dm7 G7 C6 Am7] C Extended Cadence + Dropback	[Am7 Dm7 G7 C6 Am7 Dm7]	[Am7 Dm7 G7 C6 Am7 Dm7 G7]
2			[Dm7] D m7 D Minor On	[Dm7 G7] C Straight Approach	[Dm7 G7 C6] C Straight Cadence	[Dm7 G7 C6 Am7] C Straight Cadence + Dropback	[Dm7 G7 C6 Am7 Dm7]	[Dm7 G7 C6 Am7 Dm7 G7]
3				[G7] G 7	[G7 C6] C Perfect Cadence	[G7 C6 Am7] C Perfect Cadence + Dropback	[G7 C6 Am7 Dm7]	[G7 C6 Am7 Dm7 G7]
4					[C6] C Major On	[C6 Am7] C On + Dropback C Major On Off VI A Minor Off On bIII	[C6 Am7 Dm7]	[C6 Am7 Dm7 G7] C POT
5						[Am7] A m7 A Minor On	C partial POT	[Am7 Dm7 G7] C Extended Approach
6							[Dm7] D m7 D Minor On	[Dm7 G7] C Straight Approach
7								[G7] G 7

The entry in row i *column* j *indicates the chord sequence represented by chords* i *through* j *in the original progression (in brackets), as well as the set of grammar non-terminals that expand to that sequence. There can be zero or more non-terminals in the set, although in the present example we have suppressed, for the sake of clarity, the non-terminals that turn out to be unused in the final analysis.*

by the CYK algorithm progressing from one superdiagonal to the next, until the top right corner is reached.

Once this table has been constructed, the minimum cost algorithm is applied to determine the parse presented to the user, which in this case consists of entries (0, 3) and (4, 7)—i.e., C POT (Plain Old Turnaround) followed by another C POT. The table exposes other potential explanations of the tune, which are rejected due to a higher computed cost. The effect of the cost-minimization algorithm is to compare the best single-brick explanations of subsequences of the chord sequence with synthetic combinations of subsequences that follow each other. For example, in updating entry (0, 3) of the table, the cost of a POT (20) is less than the other two possible pairings of subsequences:

(0, 0) (1, 3): Major On, Extended Approach (550 + 45 > 20)
(0, 1) (2, 3): On + Dropback, Straight Approach (550 + 45 > 20)

The POTs in (0, 3) and (4, 7) form the final sequence in entry (0, 7), with a cost of 40.

Appendix 2: Tunes Analyzed

The following lists the tunes that were evaluated, grouped according to rating. All titles are from Elliott (2009).

Rating +1: *Blue Bossa*; *Everything Happens to Me*; *Flamingo*

Rating 0: *A Fine Romance*; *A Foggy Day*; *Ain't Misbehavin'*; *Airegin*; *All of Me*; *All of You*; *All the Things You Are*; *Alone Together*; *Angel Eyes*; *Autumn Leaves*; *Avalon*; *Baubles, Bangles, and Beads*; *Bemsha Swing*; *Blue Moon*; *Blues No. 2*; *Broadway*; *Bye Bye Blackbird*; *Cherokee*; *Come Rain or Come Shine*; *Confirmation*; *Danny Boy*; *Don't Go to Strangers*; *Fly Me to the Moon*; *Honeysuckle Rose*; *On Green Dolphin Street*; *Take the 'A' Train*; *The Girl from Ipanema*

Rating−1: *A Dream is a Wish Your Heart Makes*; *A House is Not a Home*; *Afternoon in Paris*; *Anything Goes*; *Beautiful Love*; *Between the Devil and the Deep Blue Sea*; *Bewitched, Bothered, and Bewildered*; *Blue Lou*; *Blues No. 1*; *Blues No. 3*; *Body and Soul*; *But Not for Me*; *Ceora*; *Cheek to Cheek*; *Cry Me a River*; *Days of Wine and Roses*; *Devil May Care*; *Don't Take Your Love from Me*; *Don't Worry 'bout Me*; *Doxy*; *Easy Living*; *Emily*; *Everything I Love*; *Falling in Love with Love*; *Groovin' High*; *Have You Met Miss Jones?*; *Here's That Rainy Day*; *I Get Along without You Very Well*; *I Got It Bad*; *I Got Rhythm*; *I Love You*; *I Loves You Porgy*; *I Remember You*; *I Thought about You*; *I Wish You Love*; *I'll Remember April*; *I've Got a Crush on You*; *I've Got the World on a String*; *I've Never Been in Love Before*

Rating−2: *Alice in Wonderland*; *Autumn in New York*; *Bernie's Tune*; *Blue and Sentimental*; *Blues No. 4*; *But Beautiful*; *Chelsea Bridge*; *Christmas Song*; *Corcovado*; *Corner Pocket*; *Dear Old Stockholm*; *Dearly Beloved*; *Desafinado*; *Donna Lee*; *Dream Dancing*; *East of the Sun*; *Embraceable You*; *Every Time We Say Goodbye*; *Four*; *Gee Baby Ain't I Good to You?*; *Georgia*; *Gone with the Wind*; *How About You?*; *I Can't Belive that You're in Love with Me*; *I Can't Get Started*; *I Fall in Love Too Easily*; *I Guess I'll Hang My Tears Out to Dry*; *I Should Care*; *I'm Confessin' that I Love You*; *I'm Old Fashioned*; *I've Grown Accustomed to Her Face*

Rating−3: *A Ghost of a Chance*; *Cute*; *I Remember Clifford*