A CONSTRAINT SATISFACTION APPROACH TO TONAL HARMONIC ANALYSIS

TIM HOFFMAN

WILLIAM P. BIRMINGHAM

ELECTRICAL ENGINEERING AND COMPUTER SCIENCE DEPARTMENT
THE UNIVERSITY OF MICHIGAN

25 JANUARY 2000

TECHNICAL REPORT

CSE-TR-397-99

Abstract

This paper gives an algorithm for harmonic analysis for a tonal composition using a constraint-satisfaction problem (CSP) model. The algorithm combines rule-based and preference-based approaches to perform harmonic analysis, taking the position that the problem can be modeled as a computational process and less of a psychological phenomenon. Using cadential patterns within the piece, it identifies tonal centers and assigns the harmonic analysis, applying preferences when necessary. A software implementation of the algorithm is presented, along with a discussion if its results.

1 INTRODUCTION

Harmonic analysis provides a theoretical framework for examining a musical composition and summarizing its tonal organization. This analysis is based upon the rules of tonal harmony, whose application results in a grammatical parsing (Roman numeral notation) of a composition; this Roman numeral analysis gives the functions of chords relative to the key area, or tonal center, in which the chords reside. Designed to analyze Western tonal music (seventeenth to nineteenth century), it provides insight into the musical language spoken by the great composers of this era.

In recent years, there have been several attempts, with various levels of success, to render portions of harmonic analysis as algorithms. By and large, these research efforts have been informal, relying on lists of rules or sketches of algorithms. They have not addressed some of the important computational issues, such as problem complexity, algorithm complexity, and generally have not provided concise descriptions of their algorithms. By addressing these issues, we gain insight into how harmonic analysis can be tackled efficiently by computers. Further, we believe that the precision necessary for computational analysis will provide musical insight as well. Lerdahl and Jackendoff's efforts in creating a formal system for tonal music analysis, for example, have lead to important musical insights (Lerdahl and Jackendoff 1983). Likewise, we focus our efforts on tonal music analysis, examining chorales written by Samuel Scheidt, a seventeenth-century composer. Scheidt's music closely follows the conventions of tonal harmony, making it excellent test material for our algorithm.

In this paper, we model harmonic analysis as a constraint-satisfaction problem (CSP). The CSP's formal framework provides a clear representation of the problem, facilitates its analysis, and encourages concise algorithm descriptions. We chose to model the problem using attributed domains to take advantage of the properties of chords, such as key, quality, and function. We borrow this technique from the more expansive Multi-Attribute Domain (MAD) CSP (Darr 1997; Darr 1998).

In addition to its expositional advantages, the CSP naturally handles constraint and preference processing. We posit, and demonstrate in this paper, that many rules used in harmonic analysis are actually analytical constraints. By modeling these "rules" as constraints, we can both simplify processing by reducing, albeit not necessarily eliminating, the number of *ad hoc* rules, and by bringing to bear powerful algorithms from the CSP literature on harmonic analysis.

Central to our approach is the identification of tonal centers in the composition. This cannot be accomplished by a simple application of a rule, as the rules for tonal harmony are not specifically stated, but are conventions drawn from centuries of musical experience. There is no music-theoretic requirement, for example, that a tonal composition must employ the tonic-dominant relationship, yet it is ubiquitous and recognized by musically trained and untrained listeners alike. Likewise, the algorithms presented here use cadences as a basis for identifying tonal centers. Using the constraint that modulations (changes in the tonal center) are suggested by cadences, our algorithm searches a piece for all cadential structures, determines which are locally the best, and then decides whether or not to modulate. After finding the tonal centers, it labels the chords in the piece with the proper Roman numeral notation.

The lack of a formal system in tonal harmonic analysis poses a problem for a computational approach. Like natural-language parsing, tonal harmonic analysis does not necessarily give a single answer; two music theorists analyzing the same composition may arrive at different analyses, yet neither of them would necessarily be considered more correct. Many of the decisions made during analysis reflect the analyst's preferences. Thus, preferences must be included in the computational system. An example of a preference is the selection of the

location of a modulation; there may be several equally valid places for the modulation, with fixing of any one relying on the analyst's preference.

We describe in this paper the CSP representation, how the types of music we are interested in analyzing map to a CSP, and the algorithms we use to perform harmonic analysis.

2 REPRESENTATION

Understanding the algorithm presented in this paper requires a basic knowledge of CSPs. A brief introduction to CSPs follows.

2.1 CSP overview

2.1.1 Definition

A CSP comprises the following elements:

- a set of *variables* $\mathbf{V} = \{V_1, \ldots, V_i, \ldots, V_n\}$.
- a *domain* for each variable, $\mathbf{D}_i = \{d_1, \ldots, d_d\}$, which is the set of values that may be assigned to variable V_i. The domains in this paper are discrete, but in general may be either discrete or continuous. Extending d_i definition to include an *attribute*; d_i.att is in the set {key, quality, function}. This is in the spirit of the more expansive MAD-CSP definition. For example, an instantiation of a variable might be V_i = <C, Maj, I>. This tuple, consisting of three attributes, constitutes one *domain element*.
- a set of *constraints*, $\mathbf{C} = \{C_{1,1,1}, \ldots, C_{i,j,k}\}$, which place restrictions on the possible values of the variables. In this paper, we are concerned only with binary constraints (although the MAD-CSP requires no such restriction). The notation $C_{i,j.k}$ refers to a constraint that has variables V_i and V_j as arguments, with k denoting a particular constraint between these variables. In this paper, the constraints are directed, although this is not a requirement.

  The modulation constraints (Table 1) are binary. The complete notation identifying an arbitrary constraint should be $C_{i,j,k}$, representing the node, constraints involving that node, and whether it is "inbound" (coming from V_{i-1})or "outbound" (going to V_{i+1}). For notational convenience, we treat them as directional. This gives each V_i (except the first and last, V_1 and V_n) three constraints associating V_i with V_{i+1}.To simplify this notation further, we refer to only the outbound constraints from V_i. In referring to constraint $C_{i,j}$, we refer to all constraints connecting V_i to V_{i+1}. Thus $C_{2,5}$ is the constraint that specifies V_3.key = V_2.key + 7. We use this notation without loss of generality, as all the modulation constraints can still be uniquely identified.

  - Constraint-propagation functions $h_{i,j}(d_i.att)$ restrict the *attributes comprising the domain* of V_i, given the possible assignment of domain values to all other arguments.
  - The precondition-evaluation function, $c_{i,j}^{pre}$, defines when constraint $C_{i,j}$ is active. If $c_{i,j}^{pre}$ is TRUE, $C_{i,j}$ is active (it restricts D_i and D_{i+1}); otherwise, it is inactive (all domain values trivially satisfy the constraint).
  - Constraint-evaluation functions, $C_{i,j}(V_i, V_{i+1})$, return TRUE if the arguments satisfy constraint $C_{i,j}$, and FALSE otherwise.
- A constraint graph, G(V,C), which depicts the relationships among the variables and constraints. The nodes are variables, and the arcs are the constraints. Further, we assume that constraints are directed arcs from V_i to V_j, without loss of generality. In fact, as we show in this paper, G(V,C) is a hypergraph that is tree shaped.

2.1.2 Properties

A solution to a CSP is an assignment of values to all the variables that simultaneously satisfies all the constraints. A solution may require a possibly combinatorial search of the domain space to find such an assignment. A benefit of using the CSP framework is that there are many heuristics for reducing the complexity of the search process.

An important method for improving the efficiency of the search, *arc-consistency* checking, ensures that for each variable, all the domain elements satisfy the constraints on that variable. This eliminates infeasible domain elements, thereby reducing the search-space size. Arc consistency is given by Equation 1.

$$\forall V_i : \{ \forall d_i \in D_i : C_{i,j}(V_i, V_{i+i}) \}$$

Equation 1

Arc consistency does not guarantee that all constraints can be satisfied simultaneously: it does not guarantee that a solution exists.

Decomposability, the strongest form of arc consistency, (also known as n-consistency) does guarantee a solution. If graph **G** is decomposable, then all assignments of the variables constitute solutions; the domains contain no values that will cause any constraint to be inconsistent. In a decomposable graph, all variable assignments constitute solutions.

Searching a graph may also involve *preferences*, which guide the search towards more preferable solutions. An example of a preference in harmonic analysis is choosing an applied function over a root function, such as choosing `V/V` instead of `II` for a chord's function.

2.1.3 Example

Figure 1 shows a constraint graph for a simple CSP. In this example, there are only two types of constraints: $C_{i,1}$[1] specifies that V_{i+1}.width = V_i.width + 1, and $C_{i,2}$ specifies that V_{i+1}.height = V_i.height. The domains of each variable are listed in tables. For example, V_1 has a domain comprising the tuples <1,7> and <5,9>.

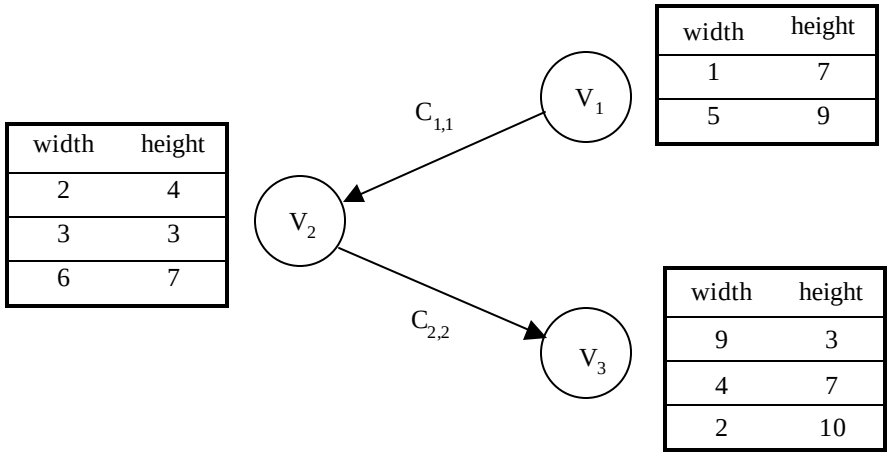

Figure 1. Initial constraint graph.

[1]To simplify the discussion, we assign a number to each constraint "type." For example, the height constraint type is always constraint 2 in the example. In general, this is not necessary, and the constraint number (the second argues of its subscript) could change.

$C_{1,1}$ specifies that V_2.width = V_1.width + 1, and $C_{2,2}$ specifies that V_3.height = V_2.height. Using arc consistency checking, we first check constraint $C_{1,1}$, and find that <3,3> can be eliminated from D_2. Assigning V_2.width = 3 violates $C_{1,1}$ because 2 is not present in the "width" column in D_1, so <3,3> is removed. Similarly, by checking $C_{2,2}$, we eliminate from D_3 the tuples <9,3> (because <3,3> is no longer in D_2) and <2,10>. After making the graph arc consistent, we are left with the diagram in Figure 2.

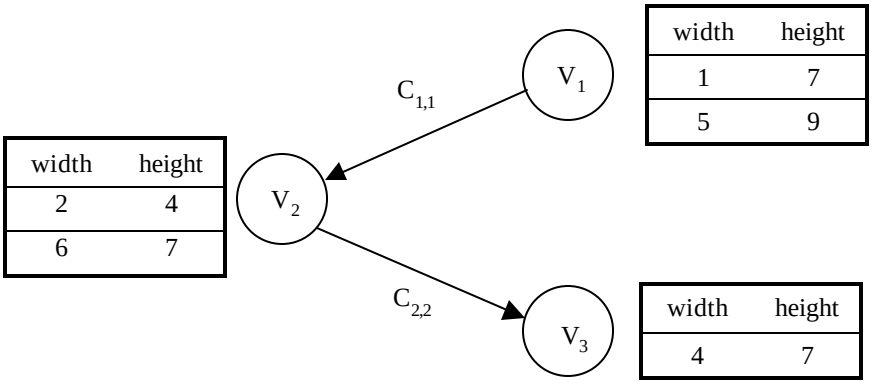

Figure 2. Arc consistent constraint graph.

The solution to this CSP is V_1 = <5,9>, V_2 = <6,7>, V_3 = <4,7>, found by instantiating variables in *reverse* order. Starting with V_3, <4,7> is the only choice. Satisfying $C_{2,2}$ forces V_2 to <6,7>, and satisfying $C_{1,1}$ forces V_1 to <5,9>. Note that the graph in Figure 2 is arc consistent but not decomposable.

2.1.4 Musical example

Suppose a sequence of three chords is analyzed. Our CSP representation has each chord a variable, whose attributes are the `key` and `quality` of the tonal area, and `function`, which is the harmonic function (Roman numeral notation). The constraints listed in Table 1 apply. Figure 3 depicts the three chords, along with its CSP representation.

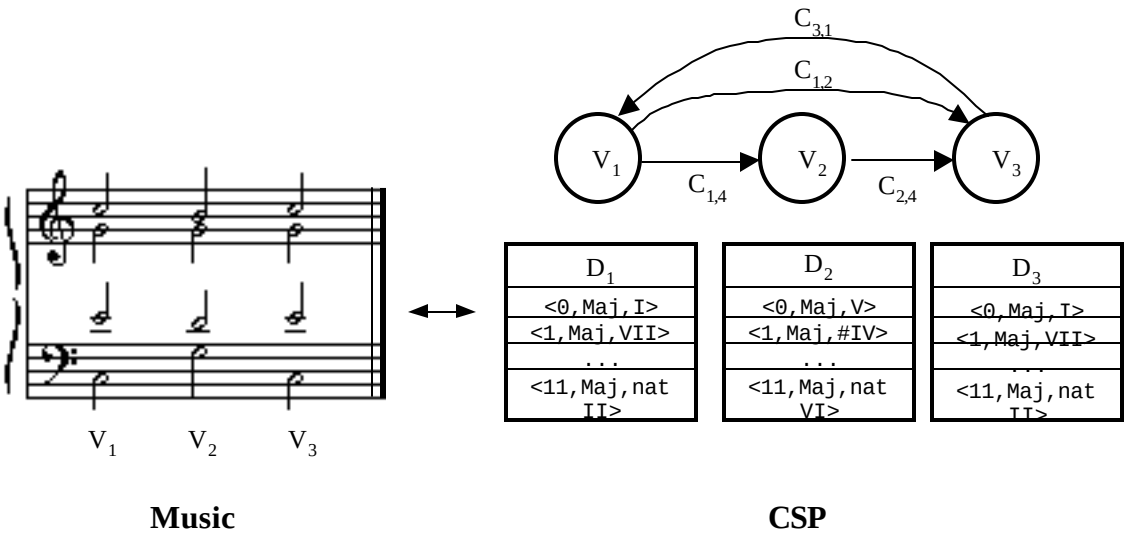

Figure 3. Music and CSP representation.

The root of V_1 is C, and by enforcing $C_{3,1}$, D_1 reduces to one element. $C_{1,4}$ similarly eliminates all but one element from D_2. (Note that pitches are assigned numbers; half steps are one increment apart.). The solution to the example in Figure 3 is the tuples <0,Maj,I>, <0,Maj,V>, <0,Maj,I>, which is a perfect authentic cadence.

2.2 CSP Musical Representation

In this section, we show the mapping from harmonic analysis (at least, the part of which we are concerned) to a CSP.

2.2.1 Assumptions

The assumptions regard the harmonic restrictions placed on the pieces we currently can analyze:

- *The key of the piece is that of the root of the first and last chords, and the quality of the key (major or minor) is that of the first chord.*

This reduces the burden of key finding in this project. This was done for two reasons: key finding is not a trivial task, with a good deal of literature devoted to this subject alone (Vos and Geenen 1996; Temperley 1997). More important, the area of interest of this algorithm is the identification of key changes within a piece. The onus of detecting modulations and key areas still rests on the algorithm, and is described in detail later.

- *All modulations are to major keys.*

This does not affect the generality of the algorithm; it simply narrowed the scope of the project. Including minor keys means having to include special cases for the different types of minor scales (melodic and harmonic). If the process can be done for major, then minor can be done in the same manner; the tables for what tones are diatonic to a given key would simply be different.

- *Modulations follow the circle of fifths.*

For example, if the current key were D Major, it can either modulate to the dominant, A Major, or to the subdominant (IV), G Major. In tonal music, the circle of fifths is not a rule, but rather a strong preference, as the tonic-dominant relationship is considered one of the strongest in tonal music (Lerdahl 1988).

- *Suspensions (a dissonance that resolves to consonance) are resolved in the input.*

Some may question the validity of eliminating suspensions (in the same manner that some object to how Schenkerian analysis removes "unnecessary" notes for its analysis), but the goal of the project is to determine different key areas of the piece. Suspensions add color and direction to the piece, but do not change the harmonic analysis.

- *Neighboring and passing tones are omitted.*

The justification for this assumption is similar to the former, as passing and harmonic tones will not change the harmonic analysis of a piece. Occasionally, a neighboring tone's motion will take it through a seventh, in which case the note is harmonically important; in these cases, the note is included.

2.3.2 Mapping

Our mapping of a composition to a CSP is in Table 1. Note that `root` is not included as an attribute of a chord. This is represented by the *function* attribute; knowing the function means knowing the root as well. In some constraints, we refer to the root of a chord; this is for notational convenience.

CSP	**Music**
Variable	Chord
Domain attributes	Key Quality (major, minor) Function (Roman numeral notation) Cadence (Boolean) Index (integer) Root Inversion Chord (array of integers that comprise the chord)
Constraint-propagation functions	1. V_1.key = V_n.root 2. V_n.key = V_1.key 3. V_i.quality = major 4. V_{i+1}.key = V_i.key 5. V_{i+1}.key = (V_i.key + 7)MOD(12) 6. V_{i+1}.key = (V_i.key +5) MOD(12)
Constraint precondition-evaluation functions (Section 3.1.4)	1. True 2. True 3. True 4. V_{i+1}.key = V_i.key 5. V_{i+1}.key = (V_i.key + 7) MOD(12) 6. V_{i+1}.key = (V_i.key + 5) MOD(12)
Search preferences	• If $\|D_i\| > 1$ and V_{i+1}.root = V_{i+1}.root+7 and V_i.key = V_{i+1}.key, prefer V_i.function = V/V to V_i.function = II • If $\|D_i\| > 1$ and V_{i+1}.root = V_{i+1}.root+1 and V_i.key = V_{i+1}.key, prefer V_i.function = vii°/V to V_i.function = #iv°

Table 1. Musical CSP representation.

The attributed domains are in the spirit of a Multi-Attribute Domain (MAD) CSP, but since the domains are treated as discrete collections of attributes, they are modeled as a traditional CSP. Of the attributes, the first three (`key`, `quality`, `function`) are directly related to music, whereas the last two (`cadence`, `index`) are particular to our algorithm.

The first three constraints (see "Constraint-propagation functions" in Table 1) establish the key of the piece. Under our assumptions, the piece begins and ends in the same key; these constraints determine the key of the first and last chords. With this information, the algorithm determines the tonal centers for the entire piece.

The last three constraints, or *modulation constraints*, contradict themselves, saying that at any time a piece simultaneously stays in the same key, modulates to the dominant, and modulates to the subdominant. This is because there are no restrictions on where modulations can occur. So, in the constraint graph, all the modulation constraints must apply to *each* node, V_i. Each modulation constraint's precondition ensures that only one modulation constraint will be active per variable.

Since the first three constraints are always active, they need no precondition-evaluation function. (Such a function would simply return TRUE.) The last three constraints, however, require precondition-evaluation functions to ensure that only one modulation constraint is active per each node. The purpose of the precondition-evaluation functions is to determine where modulations occur.

Search preferences ensure a unique solution. After arc consistency is enforced, the graphs become decomposable; however, some domains have more than one element, allowing more than one possible solution. The search preferences select the best value for the variable rather than enumerate all the possible solutions.

2.2.2 Conversion

The conversion of a musical score to the input for the algorithm is done by hand, similar to Winograd's method (Winograd 1968). The resulting list is a series of four-voice chords (listed as Bass, Tenor, Alto, Soprano), with the numbers referring to the MIDI pitch values. (For example, pitch = 60 is C_4, or middle C.) A value of –1 in a voice means it is silent in that chord. The following example shows the conversion of the chorale *Von Himmel Hoch* (a) , using the assumptions given in Section 2.2.1.

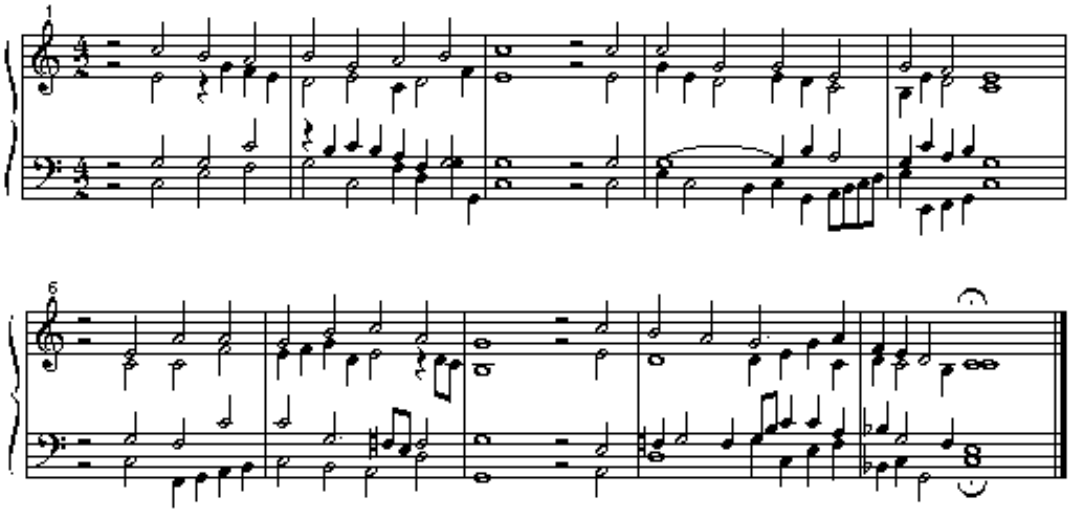

Figure 4: Original score, *Von Himmel Hoch (1)***.**

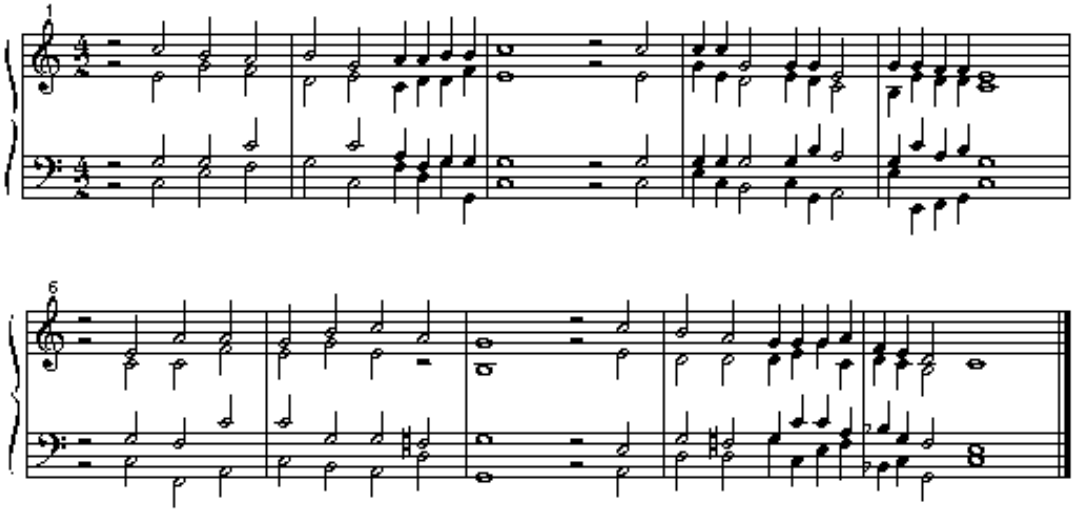

Figure 5: Chords, *Von Himmel Hoch (1)***.**

B T A S	B T A S	B T A S	B T A S	B T A S	B T A S
1) 48 55 64 72	8) 55 55 62 71	15) 48 55 64 67	22) 48 55 60 64	29) 50 54 -1 69	36) 52 60 67 67
2) 52 55 -1 71	9) 43 55 65 71	16) 43 59 62 67	23) 48 55 60 64	30) 43 55 59 67	37) 53 57 60 69
3) 53 60 65 69	10) 48 55 64 72	17) 45 57 60 64	24) 41 53 60 69	31) 45 52 64 72	38) 46 58 62 65
4) 55 -1 62 71	11) 48 55 64 72	18) 52 55 59 67	25) 45 60 65 69	32) 50 55 62 71	39) 48 55 60 64
5) 48 60 64 67	12) 52 55 67 72	19) 40 60 64 67	26) 48 60 64 67	33) 50 54 62 69	40) 43 53 59 62
6) 53 57 60 69	13) 48 55 64 72	20) 41 57 62 65	27) 47 55 67 71	34) 55 55 62 67	41) 48 52 60 60
7) 50 53 62 69	14) 47 55 62 67	21) 43 59 62 65	28) 45 55 64 72	35) 48 60 64 67	

Figure 6: Chord list.

Note that this representation disregards metrical information. This does not mean that such information is unimportant to harmonic analysis. Our aim is to identify tonal centers, and there is strong evidence that vertical sonority (chords) can provide the necessary information to do this (Cook 1987; Lerdahl 1988).

3 ALGORITHM

Central to the algorithm is the identification of tonal centers. The algorithm first finds the root and sets the domain for each chord. Based on the assumptions in Section 2.3.1, it determines the key of the piece. Next, it searches the chords for cadential patterns, creating a cadence list. This list is used to find modulations.

Cadences are central to our algorithm, as they form the basis for identifying tonal centers. In our notation, a "cadence in C" means that the root of the cadence's last chord is C. This is also the chord used to identify the cadence, allowing us to represent cadences as a subset of the set of chords. This subset is known as the *cadence space*, **Z**, where $\mathbf{Z} \subset \mathbf{V}$: it comprises a distinguished set of variables, such that each $Z_j \in \mathbf{Z}$ is the last chord of the cadence. Further, for simplicity, we number each Z_j sequentially. Note, however, that a set of chords (V_i) of arbitrary number can exist between Z_j and Z_{j+1}. The `"cadence"` domain attribute indicates if the chord is a member of the cadence space. The `"index"` domain attribute is simply the number assigned to the chord as it read from the input file.

Figure 7shows the relationship between chord space and cadence space.

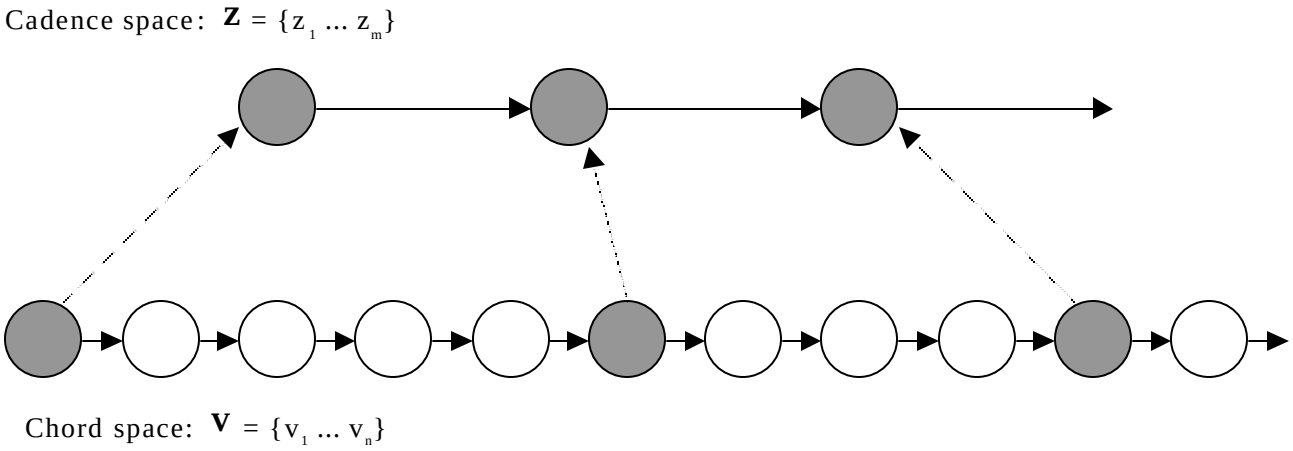

Figure 7. Cadence space vs. Chord space

It is important to note that a cadence chord (shaded) differs from a non-cadence chord only by the value of a Boolean attribute, denoted by the "Cadence" attribute in Table 1. Cadence chords simply have this flag set to `TRUE`. The reason for creating two separate chord spaces is that some of our algorithms search the chord space, while others search the cadence space. Throughout this paper, "Z_k" refers to a chord in cadence space, whereas "V_i" refers to a chord in chord space.

Using the cadences as signposts for tonal centers, the algorithm determines where modulations occur. With the tonal centers' locations defined, the preconditions for the constraints given in Table 1 come into effect. For each chord within a given tonal center, the active constraint is the one that specifies no modulation. (Constraint C_4 in Table 1.) At a modulation, the dominant or subdominant modulation constraint (C_5 or C_6) is active. The domain of each chord is searched for the elements that satisfy the constraints of key and quality.

After applying arc consistency, a search finds the solution, which is the harmonic analysis. Most of the domains reduce to a single element, but when an applied chord is possible, there will be two elements. In this case, the search preferences determine which element is the better choice.

The overall algorithm is given in Figure 8. In all of our pseudo-code listings, **V** and **Z** are represented by the variables `Chord_List` and `Cadence_List`, respectively.

```
1.  Create Chord_List from input file
2.  for i ← 1 to |V|
3.    FIND_ROOT(V_i)
4.    CREATE_DOMAIN(V_i)
5.  Cadence_List ← ∅
6.  append V_1 to Cadence_List
7.  for i ← 1 to |V|
8.    if IS_CADENCE(V_i)
9.      for j ← 1 to |D_i|
10.       v_i,j.cadence = TRUE
11.       append V_i to Cadence_List
12. mod_index ← 0
13.  for k ←1 to |Z|-1
14.   mod_index ← MODULATE(Z_k,Z_k+1)
15.   EVALUATE_PRECONDITIONS(Z_k,Z_k+1, mod_index)
16. for i ← 1 to |V|
17.   for j ← 4 to 6
18.     if C_i,j^pre = TRUE
19.       for d ← 1 to |D_i|
20.         if v_i,d.key violates C_i,j
21.           remove v_i,d from D_i
22. for i ← 1 to |V|
23.   if |D_i| > 1
24.     print preferred v_i
25.   else
26.     print v_i
```

Figure 8. Overall algorithm.

The overall algorithm is not a complete description; some sections summarize subroutines that would be unwieldy to list here. These subroutines are capitalized, as in `FIND_ROOT(`V_i`)` (Line 3.) We explain these subroutines in detail in the following discussion. Section 3.1 explains the logic for the process, and Section 3.2 analyzes its complexity. Note that the heading for each sub-routine gives the corresponding lines in the overall algorithm where that subroutine occurs.

3.1 Operation

3.1.1 `FIND_ROOT(`V_i`)`*: line 3 of overall algorithm*

The method for finding roots is similar to Hindemith's method (Hindemith 1970). The algorithm *triadifies* a chord, taking its voicing, which may span many octaves, and lowers the octaves of the tenor, alto, and soprano voices so that all are within an octave of the bass. For example, the chord G_4-C_5-E_6-C_7 (an widely spaced chord) would become G_4-C_5-E_5-C_5. Since the C_5 in the soprano is now redundant, it is not added to the 'triadified' chord, thus leaving G_4-C_5-E_5. The chord is now assumed to be in root position ('zeroth' inversion), and the voices are compared individually to the bass to determine the chord's true inversion. This is done by finding the interval between the bass and the voice being compared, and then applying rules, based on that interval, that determine whether the voice should remain where it is or be lowered by an octave. This octave lowering means that the bass note is no longer the root of the chord, and thus is of some inversion. The algorithm is given in Figure 9.

```
1.  Vi.inversion ← 0
2.    for j ← 1 to 3
3.      interval = Vi.chord[j] – Vi.chord[0]
4.      if (interval = 8) OR (interval = 9)
5.        if Vi.inversion < 1
6.          Vi.inversion ← 1
7.        lower Vi.chord[j] by an octave
8.      else if (interval = 5) OR (interval = 6)
9.        if Vi.inversion < 2
10.         Vi.inversion ← 2
11.       lower Vi.chord[j] by an octave
12.     else
13.       if interval = 2
14.         Vi.inversion ← 3
15.       lower Vi.chord[j] by an octave
```

Figure 9. `FIND_ROOT(Vi)` algorithm.

Given G_4-C_5-E_5, the algorithm first checks the interval between the first and second notes, G_4-C_5. Since this interval is a perfect fourth (corresponding to `interval` = 5 in the root-finding algorithm), `inversion` is set to two (since inversion is currently zero) by line 11, and C_5 is dropped an octave to a C_4 (line 12). Next, the interval between the first and third notes, G_4 and E_5, is found to be a sixth (corresponding to `interval` = 9); since `inversion` = 2 , line 7 does not apply, but the note is still dropped an octave (line 8). The resulting triad is now C_4-E_4-G_4. The chord is C major chord in second inversion.

3.1.2 `CREATE_DOMAIN(Vi)`*: line 4 of overall algorithm*

After triadifying a chord, a table is filled with the chord's possible functions in the different key signatures. For example, the C-major triad's functions from the example in Section 3.1.1are given in Table 2:

Chord V_i	Domain Attributes							
	Key	Quality	Function	Cadence	Index	Root	Inversion	Chord
$v_{i,1}$	C	Maj	I	FALSE	i	0	2	[60 64 67]
$v_{i,2}$	C#/Db	Maj	nat. VII	FALSE	i	0	2	[60 64 67]
$v_{i,3}$	D	Maj	nat. VII	FALSE	i	0	2	[60 64 67]
…	…	…	…	…	…	…	…	…
$v_{i,12}$	B	Maj	nat. II	FALSE	i	0	2	[60 64 67]

Table 2. Domain-attribute values for a C Major chord.

Similar tables are created for chords that are minor and diminished. (Note that all the columns to the right of "`function`" have the same value. These are properties of the chord itself and do not change, regardless of the analysis.) Initially, all the `cadence` fields are set to FALSE. The `index` field is set by a counter that increments as each chord is read from the input file. It is important to store this value, so that we can access a particular Z_i from the Cadence Space. (i.e., to determine the corresponding V_i.) Applied chords make it possible to have domain elements with the same `key` and `quality`, but different functions, as functions depend on the situation in which they are found. For example, a II chord may also be a V/V; thus, both possibilities are represented in the domains of every major chord.

Some properties of chords, such as inversion, are represented as attributes even though their values are constant throughout the domain. E_4-G_4-C_5 is a C Major triad in first inversion, regardless of the tonal center in which it is found. (Note that the triad itself has a key and quality, which are not to be confused with the `key` and `quality` of a tonal center to which the chord belongs.) The key area to which the chord belongs determines a chord's harmonic function.

3.1.3 `IS_CADENCE(`V_i`)`*: line 8 of overall algorithm*

Creating the cadence list requires identifying every cadential structure in the piece. The cadence-finding algorithm searches for three types of cadences: I-V-I, II-V-I, and IV-V-I. Plagal and half cadences (I-IV-I and V-I, respectively) are not included because they are weaker indicators of tonal change.

```
1.  elaboration ← 2
2.  j ← Vi.index – 1
3.  if ((Vj.root+7)MOD(12) = Vi.root) AND (j > 0)
4.    while elaboration > 0
5.      if Vj-1.root = Vj.root
6.        j ← j-1
7.      else
8.        break
9.      elaboration ← elaboration - 1
10. if { Vj-1.root = Vi.root OR
11.     (Vj-1.root+5)MOD(12) = Vi.root OR
12.     (Vj.root+2)MOD(12) = Vi.root  }
13.   return TRUE
14. else
15.   return FALSE
```

Figure 10. `IS_CADENCE(`V_i`)` algorithm.

The `elaboration` variable (lines 1, 4, and 9) allows for elaboration of the dominant in the cadences. For example, this means that `I-V-V`^{7}`-I`, as well as `I-V-I`, are both perfect authentic cadences. Setting `elaboration ← 2` (line 1) is a value, determined experimentally, that identifies elaborated dominants in cadences, but does not include lengthy patterns. For example, if `elaboration ← 4`, then the sequence `I-V-V-V-V-I` would be a cadence, but harmonically it is unlikely that this is the case. Line 3 checks the previous chord in the list for a possible `V-I` relationship. After allowing for elaboration of the dominant (lines 4-8), it checks the preceding chord in the list; if the root of this chord is a `II`, `IV`, or `V` in relation to the original chord (V_i), then the cadence is appended to **Z**. **Z** is stored as an ordered list; this is required for the algorithm to function properly. Since the list of chords is from which **Z** is created is ordered, **Z** does not require sorting.

The resulting list **Z** of cadence chords, $\mathbf{Z} \subset \mathbf{V}$, gives the possibilities for the tonal centers' locations within the piece, with a modulation possible between any two adjacent cadences in different keys. The relationship between the number of cadences and n, the number of chords, is not unique; there are no hard and fast rules about where cadences occur, with the exception of the end of a piece.

When the cadence-identification algorithm finishes, the list contains many 'false' cadences, whose root motions fit the cadence pattern matching, but are not true cadences in their harmonic function. The following example shows all the cadences found in a chorale by Samuel Scheidt (*Von Himmel Hoch (2)*.)

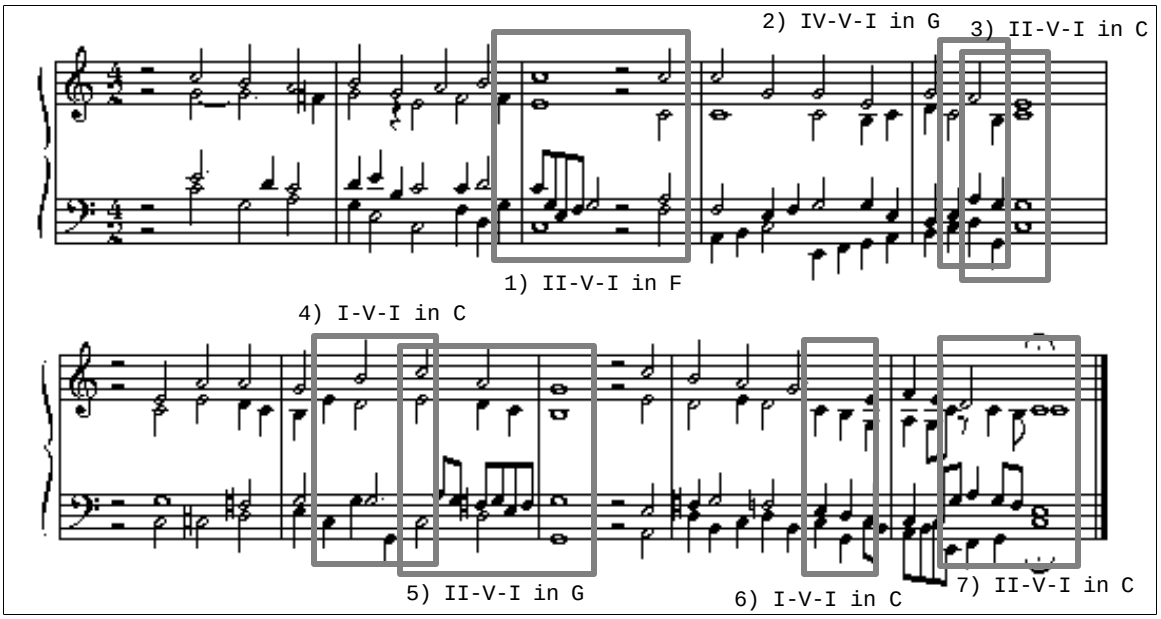

Figure 11. Cadence identification.

False cadences are accepted or rejected based on the circle of fifths (e.g., a cadence in D cannot follow a cadence in C), or by a heuristic that decides how well the cadence fits the key it suggests. For example, a G-major cadence whose chords contain no leading tones (F#) is considered much weaker than one that does contain leading tones.

In the case of overlapping cadences, this heuristic decides which is the false cadence. The criteria for elimination is the number of non-harmonic tones (NHTs), relative to the keys of the cadences, are present in the chords; the cadence whose key has the most NHTs is eliminated. Figure 12 shows an example of a D major cadence overlapping a G major cadence.

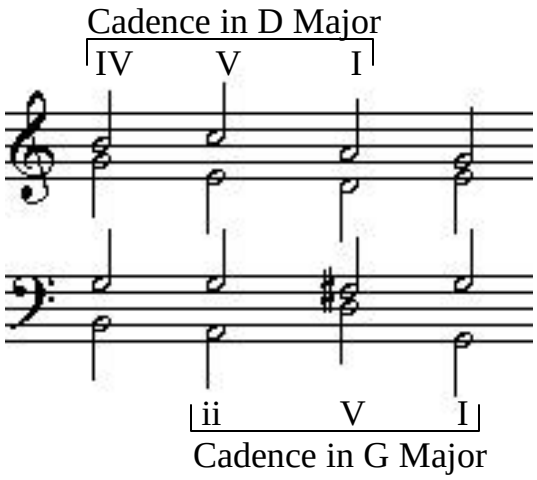

Figure 12. Overlapping cadences.

The G major cadence is chosen in this case, because the F# in the tenor voice of its second chord is a leading tone for G major. In the case of the D major cadence, its second chord has a C natural in the soprano voice, which is not a leading tone to D major. Since the preparation of the D major chord is weaker than the

preparation of the G major chord, the latter cadence is chosen as the correct one, and the former is discarded. A situation in which overlapping cadences are not resolved is when both cadences are in the same key and share only one chord. Thus, a II-V-I and a I-V-I may overlap (i.e., II-V-{I}-V-I) if the roots of the I chords are the same. In this case, both cadences are added to the cadence space.

Another criterion for cadence elimination is lack of proper voice leading. Cadences are strengthened harmonically by contrary motion between the soprano and bass and by stepwise motion in the soprano voice (Aldwell and Schachter 1978). We define the latter only between the last two chords of a cadence. In Figure 12, the cadence in D major does not have stepwise motion in the soprano voice in its last two chords. Cadences that lack both contrary motion and stepwise soprano motion are removed, as it is unlikely that they function as cadences in the piece.

Figure 13 shows the cadences that remain in *Von Himmel Hoch (2)* after eliminating false cadences.

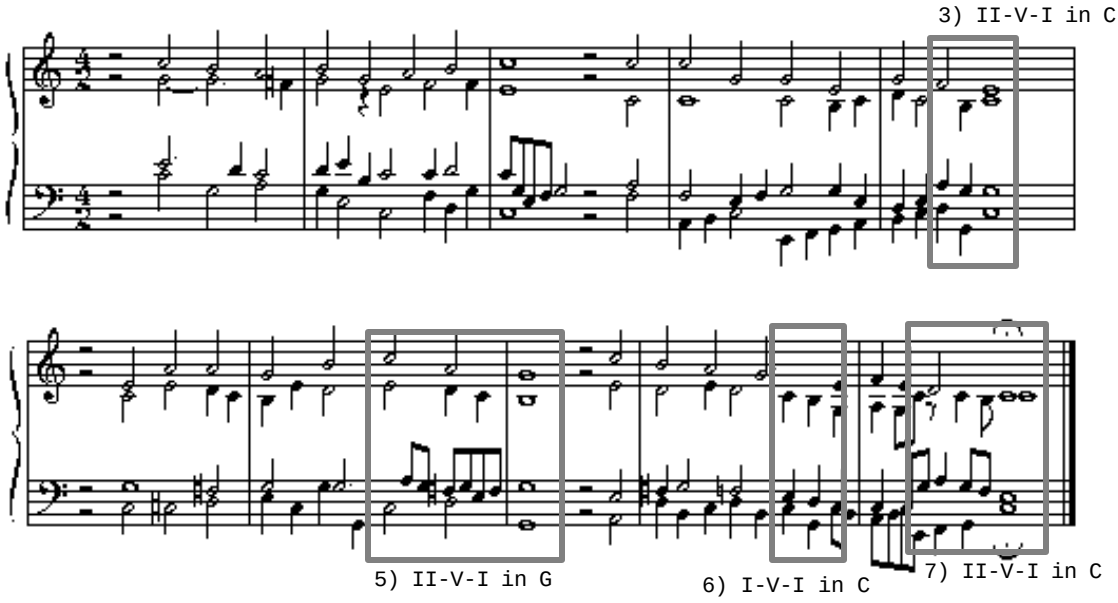

Figure 13. Final cadence list.

Cadence 1 (see Figure 11) is eliminated because of the B natural in the soprano voice, which is not diatonic to the key of F. Cadence 2 is removed by the same rule. Cadence 4 overlaps Cadence 5, and is eliminated by comparing the NHTs of the two keys. Thus, the cadences in Figure 13 comprise the cadence space.

Note that the first chord of the piece is automatically the first chord in the cadence list. This serves as the starting point for identifying modulations, which we discuss later. The cadence list from Figure 13 would be {C,C,G,C,C}. (These are the roots of the chords that comprise the Cadence Space.)

3.1.4 **MODULATE(Z_k, Z_{k+1})***: line 14 of overall algorithm*

Between sequential cadences Z_k and Z_{k+1}, a modulation is possible if their `key` attributes are different. To determine if a modulation occurs, the algorithm builds a table of the sums of all the harmonic tones, relative to all twelve major keys, of all the chords between consecutive cadences. (This is an **O**(|**V**|) operation.) To determine if the section between Z_k and Z_{k+1} modulates, the `MODULATE` algorithm compares the table values for the corresponding section for Z_k.`key` and Z_{k+1}.`key`. If the number of harmonic tonesfor Z_k.`key` is higher than that for Z_{k+1}.`key`, then no modulation occurs; the tonal center of the entire section is Z_k.`key`.

The harmonic-tone table uses a weighting to represent accurately the tonality of a section. Since a leading tone to the dominant (which is really a non-harmonic tone relative to the tonic) can still suggest the tonic itself, it is considered a *partial* harmonic tone. The weighting used in our algorithm is that a harmonic tone (relative to the tonic) has a weight of 5, whereas a leading tone to the dominant gets a weight of 1. The result is that the

presence of an occasional leading tone to the dominant in a passage does not necessarily suggest that the tonal center of that passage is the dominant; it may still be the tonic. For example, consider the following notes: C,D,E,F,F#,G. In C major, all the notes are harmonic tones except for the F#, which is a leading tone to the dominant , G major. Thus, the value for this passage in the harmonic tone table would be 26. (Five harmonic tones, one leading tone to the dominant.) In G major, the harmonic tone value for this passage would be 25; there are five harmonic tones (each with a weight of 5), and one non-harmonic tone, (F), which has a weight of zero. This passage is therefore more likely to be in C major than in G major.

If, however, the section has more harmonic tones when Z_{k+1}.key is the tonal center, then at some chord between Z_k and Z_{k+1}, the section modulates from Z_k.key to Z_{k+1}.key. A linear search of the chords between Z_k and Z_{k+1} finds the chord where the modulation occurs. If the modulation is to the dominant (Z_{k+1}.key = Z_k.key + 7), then the search order through V_i is from Z_k to Z_{k+1}. If the modulation is to the subdominant (Z_{k+1}.key = Z_k.key - 7), then a reverse search (Z_{k+1} to Z_k) is used. In both cases, the search looks for the first chord that contains a leading tone to Z_{k+1}.key. Having found this chord, the search proceeds forward (regardless of its initial direction) for the first tonic (I) chord to Z_{k+1}.key. This is the chord where the modulation occurs. In the case of a "down" modulation (V-I), it searches for the *pivot chord*, which will have both V/I and I/V functions in the two keys.

This procedure is the MODULATE algorithm. Its arguments are Z_k and Z_{k+1}, and it returns the chord number (V_i.index) where the modulation occurs. If Z_{k+1}.key = Z_k.key, then no modulation exists between the cadences, and MODULATE returns zero. A special case of MODULATE occurs when checking the last two cadences in the list. Since the piece must return to its home key (which was determined by the first three constraints in Table 1), it will always return a modulation in this case. It ignores the harmonic tone table lookup and performs the search previously described. For example, if the keys of the last two cadences are {E,A}, then it must modulate to A.

In Figure 13, modulations are possible between Cadence 3 and Cadence 5 (C-G), and between Cadence 5 and Cadence 6 (G-C). MODULATE identifies the following modulations:

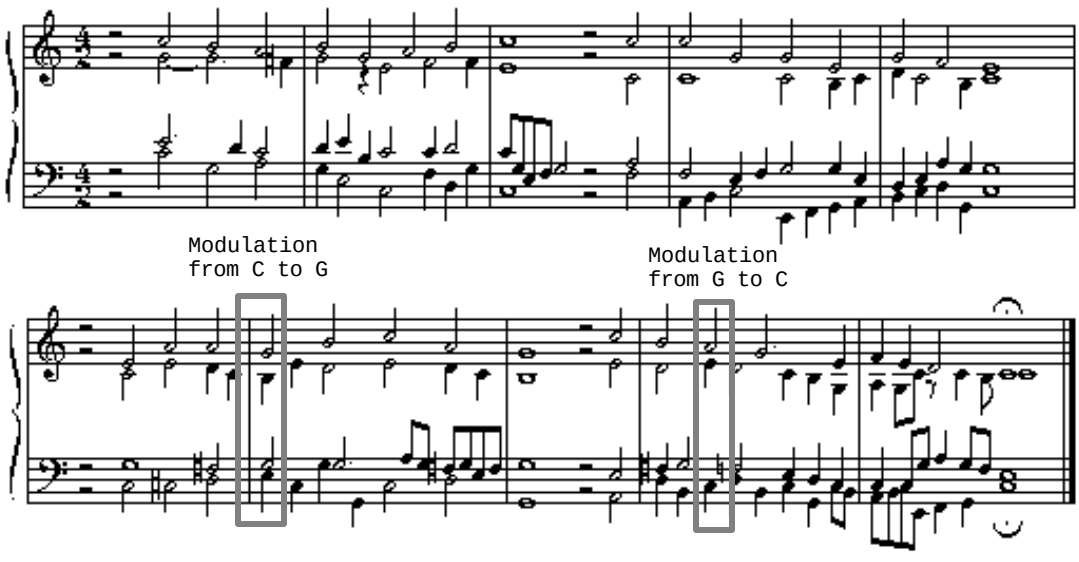

Figure 14. Modulations.

In finding the modulation from G to C (dominant to tonic), it finds the first chord with a leading tone to G by a *reverse* search, and then proceeds forward to find the first chord that serves as both a V/I in G and I/V in C. This is the pivot chord, and the modulation occurs at the following chord.

3.1.5 **EVALUATE_PRECONDITIONS(Z_k,Z_{k+1}, mod_index)***: line 15 of overall algorithm*

Table 1 lists six constraints. The first three determine the key of the piece: the first and second examine the first and last chord to get the key, and the third constraint enforces the assumption that all keys are major. Enforcing these constraints is simple, both conceptually and computationally. Applying arc consistency using these constraints affects only the first and last chords of the piece. For example, if a chorale in G major were the input, arc consistency using the first three constraints would result in everything being eliminated from D_1 and D_n except those domain elements whose `key` attribute is "7" (the key of G) and `quality` attribute is "Maj." Since there is no additional information about the tonal centers of the piece, the domains of all other variables are unchanged.

Figure 15. EVALUATE_PRECONDITIONS(Z_k,Z_{k+1}, mod_index) algorithm.

The remaining three constraints establish tonal centers. Whereas these constraints specify attributes from only two chords in its argument list, their preconditions are more complex. To determine which of these constraints is active, the preconditions are evaluated to find the modulations. (`MODULATE` is defined in Section 3.1.4.) The precondition evaluation algorithm is given in Figure 15.

If the keys of the two cadences are the same, then there is no modulation between them. In this case, lines 2-5 set $C_{i,1}^{pre}$ to `TRUE` and the other preconditions to `FALSE` for all $C_{i,j}$, where Z_{k+1}.`index` $\geq i \geq Z_k$.`index`. If the keys of Z_k and Z_{k+1} are different, then a modulation is possible between these cadence chords. If `MODLUATE` finds a modulation, then lines 6-25 evaluate the preconditions such that up to the modulation, the tonal center is that of Z_k (lines 14-17); after the modulation, the tonal center is that of Z_{k+1}(lines 22-25). Lines 14-21 set the preconditions for the modulation chord. If it modulates to the dominant, lines 14-17 apply. For modulations to the subdominant, lines 18-21 apply.

In some cases, the precondition-evaluation algorithm removes a cadence from **Z** (lines 8, 27). If `MODULATE` decides against a modulation between Z_k and Z_{k+1}, then Z_{k+1} is removed, as it is no longer an indicator of tonal change (line 8). The next comparison would be between Z_k and Z_{k+2}. If Z_{k+2} violates the circle of fifths constraint, it is also removed (line 27).

For example, suppose the following five cadences comprise the list: (C,G,D,G,C). Modulations are possible between each successive cadence pair. Suppose `Modulate` examines the first and second cadences and finds no modulation, then the second cadence (in G) is removed from further consideration, leaving the list (C,D,G,C). Now the first and second cadences (C,D) violate the circle-of-fifths-modulation constraint, so again the second cadence is removed, leaving (C,G,C).

Due to the strong tonic-dominant relationship in the pieces analyzed, this section of the algorithm also considers adjacent cadences in the dominant key to be an indication of a modulation, so the first of the two dominant cadences is a "Sforced" modulation. Thus, in a cadence sequence such as (C,G,G,C), the presence of two consecutive cadences in the dominant key (G) is considered to be a strong enough indication that a modulation occurs between the first two cadences in the list.

The result of the precondition-evaluation algorithm is the activation of the constraints that establish the tonal centers. The arc-consistency algorithm (lines 16-21 in the overall algorithm) performs a forward traversal of the chord space, removing infeasible domain elements from each chord. Due to the tree structure of the constraint graph, we do not need to recheck a domain after enforcing a constraint. Since the constraints are all binary and directed, it is impossible for constraint $C_{i,j}$ to affect the domain of any variable other than V_{i+1}.

In Figure 14, the algorithm starts at the first chord (V_1) and makes $C_{i,4}^{pre}$ `TRUE` until it reaches the first modulation (C to G, measure 7). For the chord prior to the modulation, it evaluates $C_{i,5}^{pre}$ to `TRUE`, activating the constraint for modulating up a fifth. Between the first and second modulations, it again evaluates all $C_{i,4}^{pre}$ to `TRUE` for all constraints on the chords between the modulations. Upon reaching the second modulation (G to C, measure 9), it evaluates $C_{i,6}^{pre}$ to `TRUE`, activating the constraint that specifies modulating down a fifth. Finally, it finishes the remaining segment of chords (from the second modulation to the end of the piece) by once more

evaluating all $C_{i,4}^{pre}$ to TRUE. This entire operation occurs in chord space. The resulting tonal centers are shown in Figure 16.

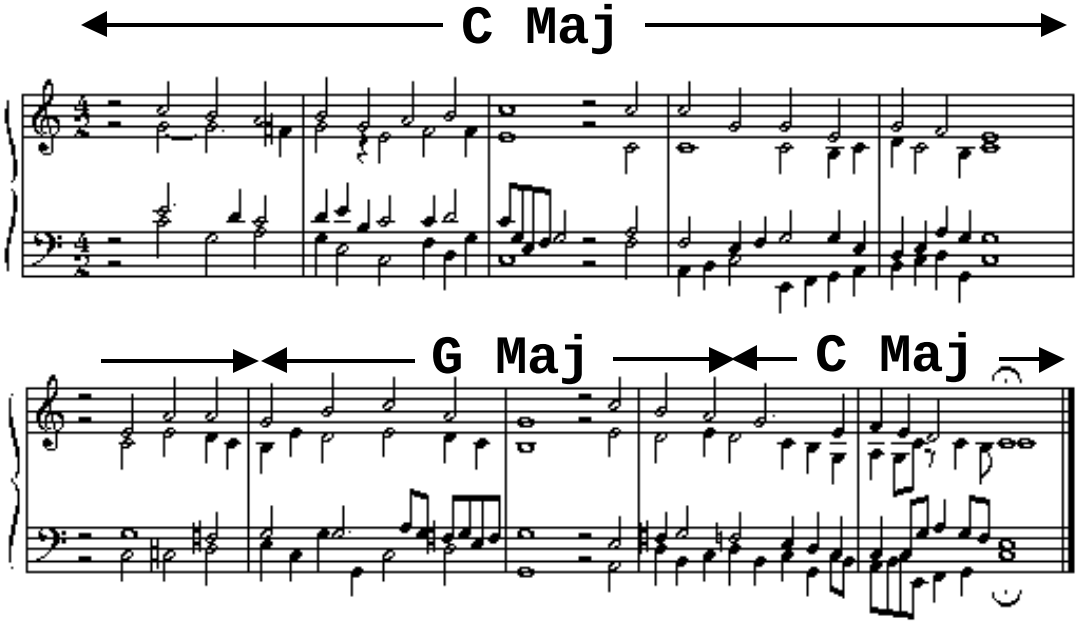

Figure 16. Tonal centers.

3.1.6 Solution

When applying arc consistency, most of the domains reduce to one element. (Recall that our definition of a domain element is a tuple of <key, quality, function>.) This makes sense, as knowing the tonal centers usually defines the harmonic functions of chords. In the case of applied or cadential chords, a domain will have more than one element, and the search preferences in Table 1 are used to choose the best analysis. Note that the graph is now decomposable; none of the domain elements remaining after G(**V**,**C**) is arc consistent will violate any constraint. The best answer is inferred by the (search) preferences, which are merely table lookups. The solution search, therefore, is linear in |**V**|, the number of chords in the piece. We prove this in the following section.

3.2 Analysis

In this section, we analyze the algorithms given in Section 3.1. By taking advantage of the CSP framework and properties of tonal music, we were able to represent harmonic analysis as a sequence of O(|**V**|) operations, as we will demonstrate in the following sections.

3.2.1 Constraint graph

The modulation constraints in Table 1 (the last three constraints) create a directed acyclic graph (DAG), shown in Figure 17. C_1 and C_2 are not shown, as they limit the domains of V_1 and V_n but nothing else; the algorithms operate under the assumption that C_1 and C_2 have already been applied. C_3 is not included in the graph, because the domains are created without any elements that violate it.

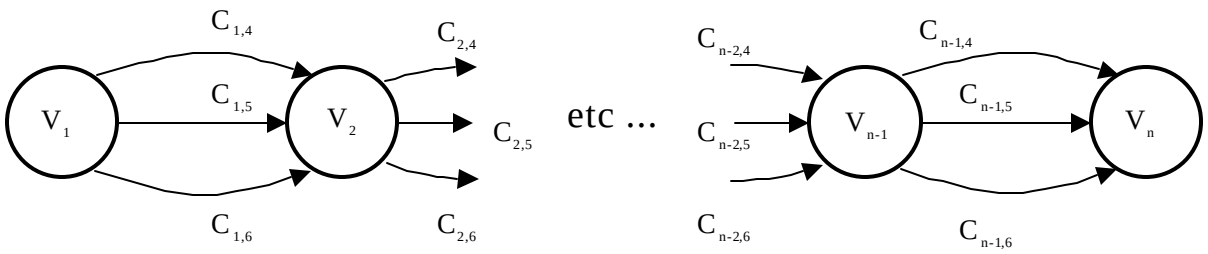

Figure 17. Constraint graph.

Without the circle of fifths modulation assumption, each node in the constraint graph would have twelve outgoing arcs instead of three, representing the possibility of modulating to any key at any time. Relaxing the major-key constraint (C_3) allows modulations to minor keys. With twelve keys and two qualities (major and minor), each node would have twenty-four outgoing arcs. The resulting graph is still a DAG. After the preconditions are evaluated, only one constraint will be active between adjacent nodes. From the perspective of a solution search, the graph is now a tree, since a search examines only the active constraints.

The structure of the constraint graph ensures a backtrack-free search. This constraint graph is ordered, meaning that the nodes are examined in a particular order (V_1 through V_n). In such a graph, the *width* of a node is the number of arcs that connect that node to its predecessors. The width of an ordering is the maximum width of all the nodes, and the width of the graph is the *minimum* width of all possible orderings. (A tree is a graph of width one.) The following theorem gives a relationship between the width of a graph and the tractability of search (Dechter 1990).

> An ordered constraint graph is backtrack-free if the level of directional strong consistency along this order is greater than the width of the ordered graph.

The algorithm given in this paper enforces directional two-consistency on an ordered graph of width one, ensuring that the search will be backtrack-free.

3.2.2 Complexity analysis

It is important to note that in a worst-case analysis, Chord Space and Cadence Space are the same magnitude. A cadence must be at least three chords long, and cadences cannot overlap. A piece with a cadence in every possible location would contain at most $|\mathbf{V}|/3$ cadences. In tonal music, however, cadences are rarely found in such close proximity.

The following analyses first investigate the subroutines, and then the overall algorithm in Figure 8.

3.2.2.1 `FIND_ROOT(Vi)` *and* `CREATE_DOMAIN(Vi)`

`FIND_ROOT` executes a constant-time loop (root finding) and then `CREATE_DOMAIN` fills a finite-length table with domain attributes. The size of this table is a constant, independent of $|\mathbf{V}|$. As both these algorithms are called once for each chord, root-finding and domain creation for the chord list requires $O(|\mathbf{V}|)$ time.

3.2.2.2 `IS_CADENCE(Vi)`

Creating the cadence space requires a linear search of the chord space. At each iteration, it performs a constant-time loop (the constant is the `elaboration` variable.) Before the cadences are appended to the cadence list, they are put temporary list that, for each cadence, stores its first and last chords. (The result is a list that is ordered by the *last* chords of the cadences.) This temporary list is traversed in a loop. Any cadences that overlap are submitted to a constant-time check (Figure 12), and a cadence gets eliminated. Cadence Space is thus created in $O(|\mathbf{V}|)$ time.

3.2.2.3 MODULATE(Z_k, Z_{k+1})

This algorithm first creates a table of the sums of all harmonic tones, relative to all twelve keys, of the chords between the cadences. Creating this table requires a traversal of chord space, with twelve constant-time operations per chord. This runs in O(|**V**|) time, and is performed only once. Although it is not specifically called by MODULATE, its analysis is included here for convenience. Next, it identifies modulations via table lookup, or removing cadences from the cadence space if it finds no modulation. Thus, creating the table requires O(|**V**|) time, and MODULATE runs in constant time. Since MODULATE is called for each sequential cadence pair (and O(|**Z**|) = O(|**V**|)), finding the modulations requires O(|**V**|) time.

3.2.2.4 EVALUATE_PRECONDITIONS(Z_k,Z_{k+1}, mod_index)

Although at first glance this algorithm appears to require O(|**V**|^2) time, its complexity is linear in |**V**|. The reason is that the main loop, which iterates through cadence space (which is O(|**V**|)). When a modulation is identified an inner loop iterates through chord space between cadence chords Z_k and Z_{k+1}. This means that cadence chords can be examined twice; once in the cadence-space search, and once in the chord-space search. At most, this will result in O(|**V**| + |**Z**|) operations. Since O(|**Z**|) is equal to O(|**V**|), the this algorithm runs in O(|**V**|) time.

Applying arc consistency also runs in O(|**V**|) time. Given the nature of the constraints, we can search through chord space, enforcing the active constraint for each V_i, which limits the domain for V_{i+1}. With the tree structure of **G,** we are assured that once a constraint is applied, it will never need to be re-checked.

3.2.2.5 Solution:

Finding a solution requires a linear search of chord space and returning the domain element for each variable. Almost all domains reduce to one element after the application of arc consistency; those that have more than one element use a constant-time table lookup (preferences) to return the best choice. This operation requires O(|**V**|) time for one solution. As demonstrated in Section 3.2.1, the search is bactrack free.

3.2.2.6 Overall algorithm:

We can now evaluate the complexity of the overall algorithm (Figure 8.) Lines 2-4 are the root-finding algorithm (O(|**V**|).) Lines 5-13 represent the cadence identification algorithm (O(|**V**|).) Lines 14-20 comprise the precondition evaluation algorithm (O(|**V**|)), followed by lines 21-26, which apply arc-consistency (O(|**V**|).) Finally, lines 27-31 find the solution (O(|**V**|).) All of the sub-algorithms execute sequentially, thus making the overall algorithm a series of linear-time operations; it therefore runs in O(|**V**|) time.

4 RESULTS

The source material analyzed here come from *Das Gorlitzer Tablaturbuch*, a collection of chorales written by Samuel Scheidt in 1650. We chose Scheidt because his work fits the assumptions we impose, but has sufficient harmonic motion to make it useful test material. A brief discussion follows each analysis.

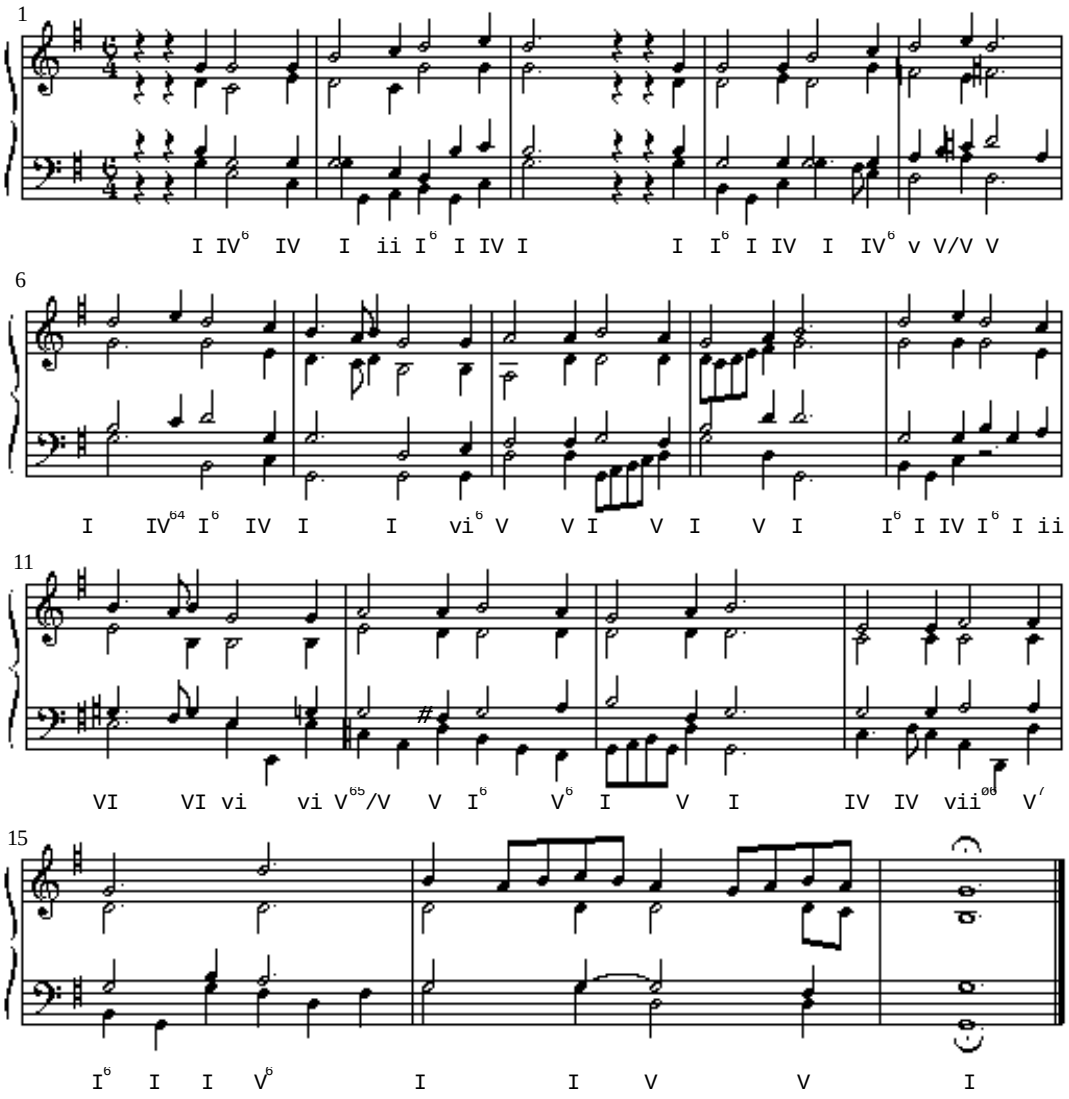

Figure 18. ***In Dulci Jubilo.***

After removing the false cadences, the key of all cadences in **Z** is G. This means that there are no modulations, and constraint $C_{i,4}$ is active for all nodes. Arc consistency thus eliminates all $v_{i,d}$, where $v_{i,d} \neq 7$ (where 7 is the pitch value of G.) The secondary functions in measures 5 and 12 are the result of preferences employed when searching for the solution.

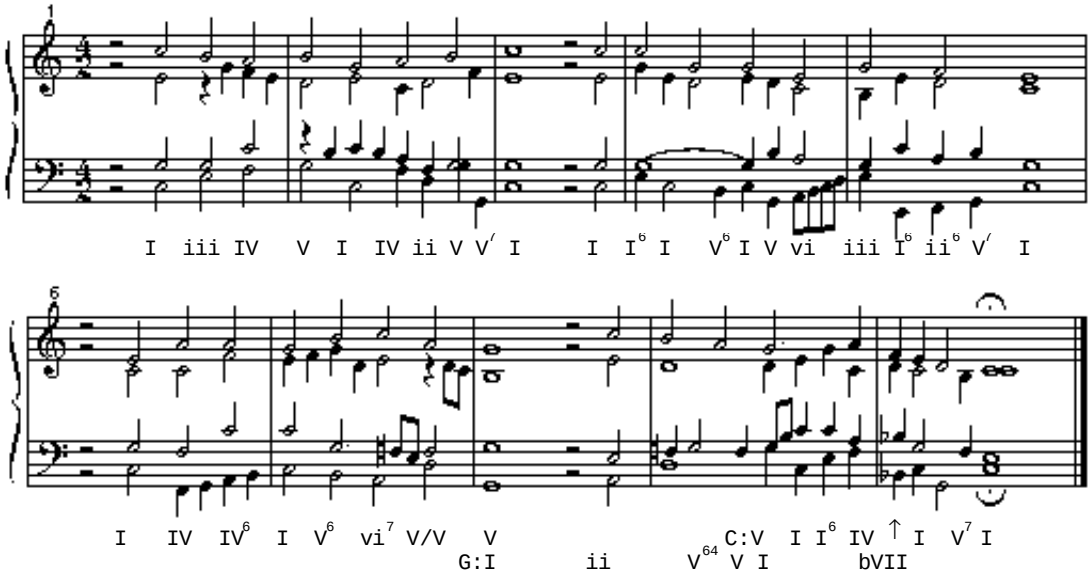

Figure 19. ***Von Himmel Hoch (1).***

In this example, **Z** contained the cadences {C,C,C,C,G,G,C} after false cadences were removed. The first of the two G cadences is at the end of measure 7. `MODULATE` examines the passage between this cadence and the C cadence at the end of measure 5, and decides that the modulation occurs at the first chord of measure 8. Since the next cadence in the list is another G cadence (measure 9), the tonal center between these two cadence chords is G. Since this is a "down" modulation, `MODULATE` looks for the last leading tone to the new key (F natural in this case, found by a *reverse* search.) From there, it looks for the pivot chord, one whose `function` is `V` in the current key and I in the new key. It assigns all current key to the chords up to the pivot chord, and the new key to the chords following the pivot.

Another chorale, *In Dulci Jubilo*, had an analysis in which the entire piece was in one key (G major) except for a small group of chords, corresponding to about a measure in the piece. In this area, the analysis showed a modulation to the dominant (D major). According to a music theorist, this is not correct, or at least very unlikely, not making any ' "musical sense." From a listener's perspective, however, it may make more sense. The analysis resulted from two D major cadences in close proximity, which might sound to the listener as if the piece briefly suggests the dominant before returning to the tonic. As resolving this is more of an interpretation rather than a rule, we do not see this result as a failing of the algorithm.

5 DISCUSSION

The goal of any theory is to represent a system with a set of concise and powerful rules. A well-formed theory summarizes and yields insights to the system, and presents a framework that others may examine and reproduce. An algorithm designed to encapsulate tonal music theory must possess these properties if it is to be a useful tool for approaching the problem.

5.1 Earlier attempts

Winograd, treating music as a grammar, developed a hierarchical set of parsing rules that act on a list of chords (Winograd 1968). Whereas he clearly enumerates the rules, hierarchies, and tables used in his implementation, the description of the implementation itself is high level, lacking necessary detail for a thorough analysis. This makes it difficult to reproduce his results.

Using a rule-based system, Maxwell represents harmonic analysis essentially as a table lookup (Maxwell 1992). In addition to the lack of a clear description of the algorithms, his presentation of the rules involved is sometimes ambiguous or missing altogether. For example, this is the description of one such rule:

> "IF a sonority is not tertian OR it is accented AND dissonant AND the next sonority is tertian AND the next sonority has a lower tertian-dissonance level OR it is unaccented AND dissonant AND the last sonority is tertian AND the last sonority has a lower tertian-dissonance level, THEN the sonority is *dissonant in context*."

Given this description, there are several possible implementations of this rule. (This is only one of over 50 such rules.) With ambiguous and omitted rule definitions, along with no description of the algorithms, it is impossible either to reproduce or to analyze Maxwell's results.

Another problem inherent with a rule-based approach is that a rule-based system requires a "proliferation of rules that tend never to reach an end" (Ebcioglu 1992). Suppose the scope of the project were to analyze just one composer, such as Bach. It would be no small task to develop the a knowledge base of all the situations in which Bach "breaks the rules" of tonal harmony, much less represent all his stylistic idiosyncrasies. Even if such a set of rules were created, it would certainly need amending to be used to analyze another composer's work. Since constraints are a type of rule, our work may also suffer from this problem as we extend it to analyze more complex music.

The Serioso program (Temperley and Sleator 1999) takes a comprehensive approach, separating its analysis into three sections, (meter, key, and harmony), and combining these separate analyses into a final interpretation. It addresses many of the difficulties encountered by the approaches described above, summarizing its performance by a set of preference rules. It introduces some intriguing ideas and insights, such as the line of fifths model (Temperley 1997).. There is no description of how the preference rules are implemented, leaving no opportunity for their analysis.

5.2 Our representation

The goal of this work is not only to model harmonic analysis,, but also to present it in a formal framework that facilitates analysis and duplication. We make no claim that our approach more comprehensive or correct than any of the others previously mentioned. Ours is an attempt to model some key insights into harmonic analysis and to present it a more accessible way.

The central idea in our approach is that cadences form a basis for identifying the presence of tonal centers within a composition. The notion of using cadences as musical guides is not new; Lerdahl, in correspondence with Jackendoff (Jackendoff 1991) suggests using cadences as aids for metrical analysis. We posit that they also play a vital role in determining the tonal structure, serving as "musical footprints" of the path the tonal center takes throughout the piece. From this, we develop heuristics to determine where cadences occur and where the tonality changes in the piece. Once the tonal centers are identified, harmonic analysis reduces to little more than a parsing problem. We, therefore, focus our algorithm on the identification of tonal centers.

We chose the CSP framework for our approach because of its elegance in representing the problem and because it is readily analyzed by formal methods. By making the variable a chord rather than the individual notes, we take advantage of properties of chords, while retaining the ability to constrain their analysis by their attributes. A major point in favor of our approach is its simplicity. Instead of using a large set of rules, we employ a series of simple pattern-matching and search algorithms to accomplish the task. We freely admit, however, that this is neither a comprehensive nor final solution to the problem. Certainly, some of the algorithms are not sufficient for analyzing all types of tonal music; our assumptions need to be relaxed, and more complicated test cases are necessary. The complexity of the algorithms may very well increase as the work progresses; the music analyzed in this paper is relatively simple, and as more complex pieces are introduced, the sophistication of the algorithms will have to improve accordingly.

6 SUMMARY

In the past, attempts at algorithms that perform harmonic analysis have suffered from a lack of formal presentation of the methods involved. Regardless of the successes of these efforts, analysis and duplication of their results is ambiguous and difficult. The purpose of this work is to encapsulate the process of harmonic analysis in a precise and well-studied framework that avails itself to analysis and reproduction.

Harmonic analysis is similar to natural-language parsing in that it incorporates rules and preferences, and like understanding natural language, is considered to be a task performed without difficulty by those who are familiar with it (Lerdahl and Jackendoff 1983). By modeling the rules of tonal harmony as constraints and the preferences as search preferences, the CSP framework provides an elegant algorithmic representation of harmonic analysis.

Our implementation employs a series of algorithms that employ various tonal harmony rules to identify tonal centers within a composition. Establishing this information is the heart of harmonic analysis, as it reduces the remainder of the problem to little more than parsing. Aiding our algorithms are several assumptions about tonal music, which simplify the task. They provide additional constraints on interpreting the compositions.

The music used to evaluate our system is a subset of tonal music, and that the assumptions we made must be relaxed in order to have a robust tonal harmonic-analysis system.

REFERENCES

Aldwell, E. and C. Schachter (1978). Harmony and Voice Leading. New York, Harcourt Brace Jovanovich, Inc.

Cook, N. (1987). "The perception of large-scale tonal closure." Music Perception **5**: 197-206.

Darr, T. (1997). A constraint-satisfaction problem computational model for distributed part-selection problems. Electrical Engineering and Computer Science Department. Ann Arbor, The University of Michigan.

Darr, T. P., W.P. Birmingham, N. Scala (1998). A MAD Approach for solving part-selection problems. Artificial Intelligence in Design, Lisbon, Portugal., Kluwer.

Dechter, R.(1990) Constraint Networks. Encyclopedia of AI**:** 276-285.

Ebcioglu, K. (1992). An Expert System for Harmonizing Chorales in the Style of J. S. Bach. understanding Music with AI: Perspectives on Music Cognition. M. Balaban, K. Ebcioglu and O. Laske. Menlo Park, AAAI Press**:** 295-333.

Hindemith, P. (1970). The Craft of Musical Composition. New York, Schott Music Corporation.

Jackendoff, R. (1991). "Musical Parsing and Musical Affect." Music Perception **9**(2): 199-230.

Lerdahl, F. (1988). "Tonal Pitch Space." Music Perception **5**(3): 315-350.

Lerdahl, F. and R. Jackendoff (1983). A Generative Theory of Tonal Music. Cambridge, Mass, MIT Press.

Maxwell, H. J. (1992). An Expert System for Harmonizing Analysis of Tonal Music. Understanding Music with AI: Perspectives on Music Cognition. M. Balaban, K. Ebcioglu and O. Laske. Menlo Park, AAAI Press**:** 335-353.

Temperley, D. (1997). "An Algorithm for Harmonic Analysis." Music Perception **15**(1): 31-68.

Temperley, D. and D. Sleator (1999). "Modeling Meter and Harmony: A Preference-Rule Approach." Computer Music Journal **23**(1): 10-27.

Vos, P. G. and E. W. V. Geenen (1996). "A Parallel-Processing Key-Finding Model." Music Perception **14**(2): 185-224.

Winograd, T. (1968). "Linguistics and the Computer Analysis of Tonal Harmony." The Journal of Music Theory **12**: 2-49.