HEURISTIC FOR MULTI–VARIABLE MUSIC STATE RECOMMENDATIONS

By

Joseph D. Valerio

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: COMPUTER SCIENCE

Examining Committee:

_____

Selmer Bringsjord, Ph.D., Thesis Adviser

_____

Michael Lynch, Member

_____

Jonas Braasch, Ph.D., D.Eng., Member

Rensselaer Polytechnic Institute
Troy, New York

April 2013
(For Graduation May 2013)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No. 1002851. SNARK theorem prover was utilized for state proofs. The parameter matrices and figures were generated in MATLAB using the results from SNARK proofs. The figures utilize the PLOTCUBE function created by Thomas Montagnon; code is distributed under the BSD license and may be found at `http://www.mathworks.com/matlabcentral/fileexchange/15161-plotcube`. In addition, the author would like to thank Dr. Bringsjord, Dr. Braasch, and Simon Ellis for their support and guidance while working on the CAIRA project.

ABSTRACT

One current challenge in the CAIRA project, which is devoted to the relationship between creativity, logic and spontaneity in musical environments, is to provide logically provable recommendations to a creative agent based upon the input of two players in an ensemble, as well as the agent itself. We report on the meeting of this challenge herein. At present, the recommendations are proven valid based upon one variable for each player, and each possible combination of player variable ranks may be quickly computed and proven. When the agent enters a state that has not yet been proven, it uses a nearest—neighbor approach to determine which state to enter. However, as more player variables are added, recommendation error and lookup time increase. The focus of this thesis is to provide a method for providing more accurate guess states to the agent in a timely manner.

1. INTRODUCTION

CAIRA, the Creative Artificially-Intuitive and Reasoning Agent, is a system in development that will be able to provide guidance to ensembles, compose music and perform in an improvisational fashion. A major goal of the CAIRA project is to develop an understanding of the relationship between logic and spontaneous creativity within the scope of musical performances. CAIRA consists of two modules that are standalone – FILTER and Handle. FILTER is the improvisational portion of CAIRA, and utilizes a combination of machine learning algorithms and Computational Auditory Scene Analysis in order to produce its own sound in response to an ensemble. Handle, the portion we are focusing on, is the logical aspect (Ellis et al., 2012).

Handle is designed with the goal of reasoning over any performance in order to function as a conductor or teacher to assist performers. In the grand scheme of things, we are forming a music calculus which will enable it to reason about not only the technical details of a piece (via music score), but the beliefs of the performers and audience as well. While there exists much work for Artificial Intelligence and music generation, approaches toward music understanding by an agent has not been investigated much (Cope, 2000, 2001, 2006, 2008). As such, the development of a music calculus that incorporates several musical models and allows us to reason about the expectations for a performance would be a major stepping stone toward a creative agent (Ellis et al., 2012).

Focusing on one of the more immediate goals of the CAIRA project, Handle is being made to reason about music state with regards to an improvisation. For this particular setup, CAIRA accepts as inputs three audio channels: two corresponding to two human players, and the third being the output of CAIRA itself. In order to determine what CAIRA plays as output, it utilizes Computational Auditory Scene Analysis (CASA) algorithms to extract features of each player's performance during a given slice of time. These features — tension, dynamic, tempo, valence and activity — are mapped to an integer scale from which the state of the performance can be

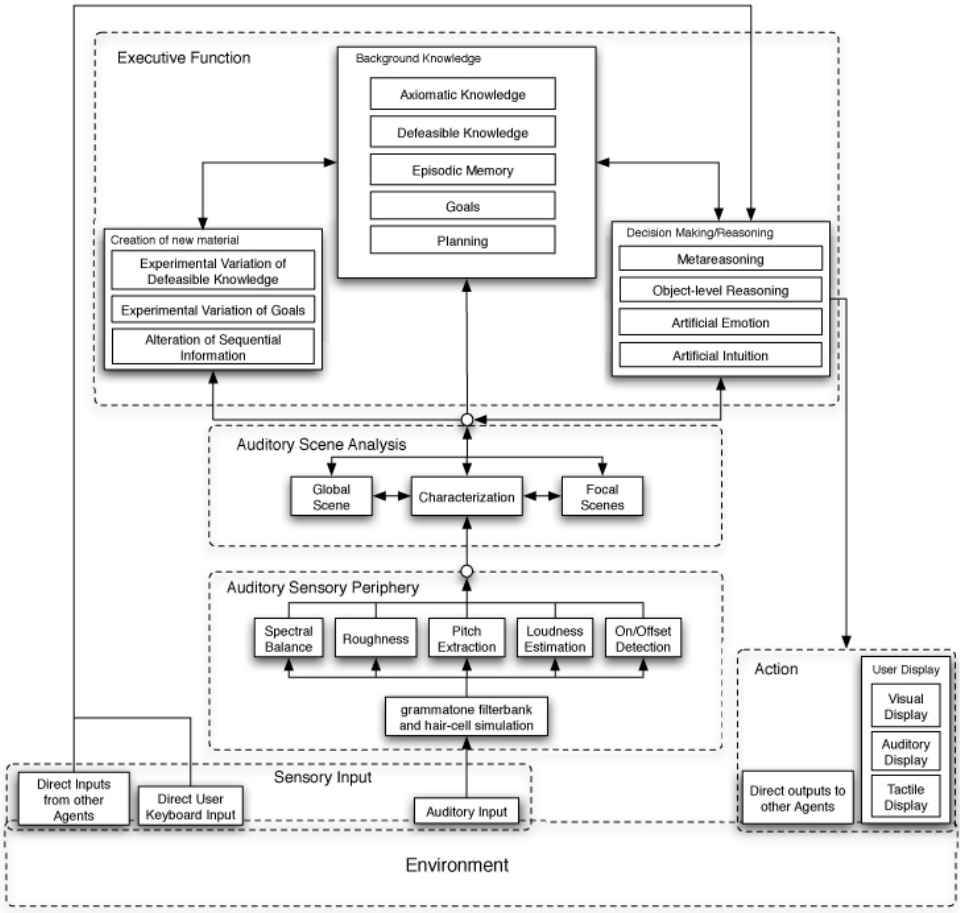

Figure 1.1: Overview of CAIRA (Ellis et al., 2012).

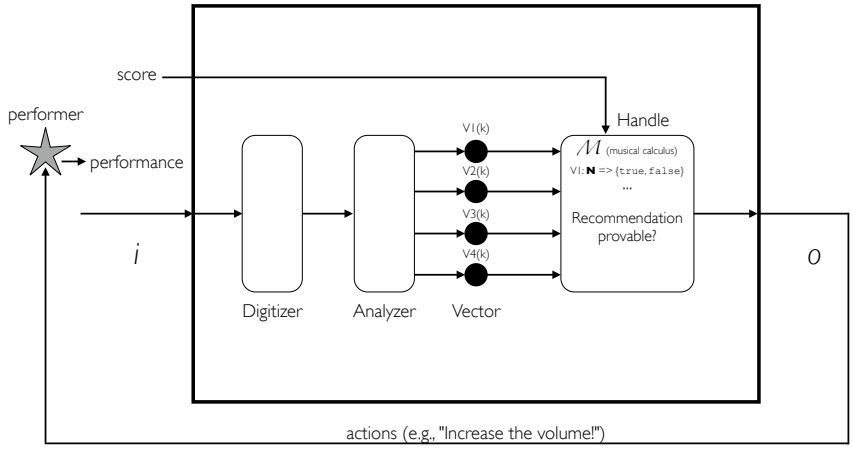

Figure 1.2: Handle Architecture (Ellis et al., 2012).

determined (Braasch, 2012). These states include player solos, low and high tension tutti, ending, and a state for uncertainty, explained in Section 2.1. From the state recommendation that Handle returns, FILTER then adjusts its own parameters such that the music it plays adheres to the recommendation.

CAIRA first attempts to look up the recommended state using the feature data from an internal data structure which consists of several three dimensional matrices of side length corresponding to the integer scale. If there is no recommendation, it uses a nearest—neighbor approach to determine the best guess for the current state. After the performance is over, the parameters are converted into inputs of the proper format for use by an automatic theorem prover, SNARK (Stickel, Stickel). Using these inputs and a series of defined rules in first—order logic (Russell and Norvig, 2003), SNARK can then prove the expected state for the parameter set.

Currently CAIRA uses one variable — $\texttt{Tension}_B$ — to determine music state, but we desire to factor in the other features for state determination. The nearest—neighbor approach to determining unknown states for one variable is a relatively quick and simple process on three dimensions with a small number of states, but as the number of input features increase, the number of states grows exponentially. When there is a small number of pre-defined data points within the cubes, searches on higher dimensions on average require large hypercubic neighborhood search regions.

> Consider a data set of size N in the d–dimensional unit hypercube, and assume hypercubic neighborhoods of side b and volume b^d.
>
> To contain k points, the average neighborhood must occupy a fraction k/N of the entire volume, which is 1. Hence, $b^d = k/N$, or $b = (k/N)^{1/d}$ (Russell and Norvig, 2003).

For our purposes, let us demonstrate the impracticality of a nearest—neighbor search using a base case in which we are given ten randomized points of the state data to begin with. For three dimensions and $k = 1$:

$b = (1/10)^{1/3} \approx 0.464$

For fifteen dimensions (five features, three players) and $k = 1$:

$b = (1/10)^{1/15} \approx 0.858$

In the latter case, the neighborhood must span nearly the entire input space in order to contain one data point on average; this is not very reliable or efficient. It is therefore the goal to come up with a heuristic such that for larger combinations of variables there is small lookup time, and the state matrix generated by the heuristic is similar to the actual recommendation matrix in a statistically significant way.

2. SINGLE VARIABLE RECOMMENDATIONS

2.1 Definitions

In order to use player inputs to logically determine what state the music is in, we must first create logical expressions which are equivalent to the definition of the state. Merriam-Webster defines solo as "a performance in which the performer has no partner or associate (Merriam-Webster, a)." However, given that CAIRA is being utilized in an improvisational manner, solo refers to "any one player's improvisation over one or more choruses of the tune (Brotman, Brotman)." A tutti is "a passage or section performed by all the performers (Merriam-Webster, d)." The end state occurs when all musicians have stopped playing. Finally, we may define the uncertain state as happening when it is not possible to conclude which of the other states the performance is in.

2.2 Logical Representation of States

Given the prior definitions and an integer value on a finite scale representing a player's calculated parameter for specific time duration, we may assign a mathematical representation to each state. We may define a solo as one player whose value is significantly greater than the other two; this produces a solo state for each player. This threshold is given as the soloist playing at least two levels above every other player. There are also two types of tutti states: low level and high level. A high level tutti is special in that it occurs in the top two levels of the scale; if any two players are in this range, there necessarily cannot be a solo. Finally, the end occurs when all players are at level 0 (Braasch, 2012). This is simply expressed in Table 2.1 on page 6.

Since we have a mathematical representation, each state may now be expressed via first—order logic (Russell and Norvig, 2003). In order to better understand the following notation, we give a brief overview of first—order logic syntax. First, we define a domain of objects to which we apply our model. These objects are constant symbols, such as the notation for a particular player, parameter level, or state. We

Table 2.1: Ensemble states as determined by parameter level. A, B, and C represent the calculated status of Musicians A, B, and CAIRA respectively. The asterisk denotes that CAIRA may disregard the recommendation and respond differently. The states are ordered hierarchically such that state overlap goes to the higher ranking state (Braasch, 2012).

Ensemble States	Musician A	Musician B	CAIRA C
Solo A	A>B+1	B+1<A	C+1<A *
Solo B	A+1<B	B>A+1	C+1<B *
Solo C	0<A<5	0<B<5	Decision Needed
Low Level Tutti	0<A<5	0<B<5	Decision Needed
High Level Tutti	A>4	B>4	C>4 *
Ending	A=0	B=0	C=0 *

may then define relations between certain objects. For instance, a player may have a certain parameter level, so we might define the relation as $Level(player, level)$. It is then possible to create sentences via logical connectives ($\wedge$ — conjunction, $\vee$ — disjunction, $\rightarrow$ — conditional, $\leftrightarrow$ — biconditional) and quantifiers ($\forall$ — universal, $\exists$ — existential). Sentences allow us to specify the meanings of relations such that they behave as we expect. If some level A is less than some level B and it is expressed as the relation $LessThan(A, B)$, $LessThan(B, A)$ is possibly true until we state that $LessThan(A, B) \rightarrow \neg LessThan(B, A)$ (If A is less than B, then B is not less than A).

Quantifiers allow us to make general statements over the domain by using variables instead of constants in conjunction with relations and sentences. If the universal quantifier ($\forall$) is used with a variable, then the sentence within the quantifier/variable pair is true for every object within the domain. Similarly, a sentence bounded by an existential quantifier ($\exists$) is true so long as there is at least one object which satisfies the sentence. Finally, if a quantifier is negated, it is equivalent to the other quantifier with the internal sentence negated. For example, if there doesn't exist an object that is red, then all objects are not red ($\neg\exists O Red(O) \Leftrightarrow \forall O \neg Red(O)$). Similarly, if not all objects are red, there exists an object that is not red ($\neg\forall O Red(O) \Leftrightarrow \exists O \neg Red(O)$). With this understanding established, we may examine the rules currently in place for one feature.

- $\forall level(\neg LessThan(level, level))$
  - A level may not be less than itself.
- $\forall level1 \forall level2(LessThan(level1, level2) \rightarrow \neg LessThan(level2, level1))$
  - If some level1 is less than level2, level2 is not less than level1.
- $\forall level1 \forall level2 \forall level3((LessThan(level1, level2) \wedge LessThan(level2, level3)) \rightarrow LessThan(level1, level3))$
  - The transitive property of inequality.
- $LessThan(a, b) \wedge LessThan(b, c) \wedge LessThan(c, d) \wedge LessThan(d, e) \wedge LessThan(e, f) \wedge LessThan(f, g)$
  - Establish ranking of level constants within our domain.
- $\forall level(WithinOne(level, level))$
  - A level is within one of itself.
- $\forall level1 \forall level2(WithinOne(level1, level2) \leftrightarrow WithinOne(level2, level1))$
  - level1 is within one of level2 if and only if level2 is within one of level1. This also establishes that level1 is not within one of level2 if and only if level2 is not within one of level1.
- $\forall level1 \forall level2 \forall level3((WithinOne(level1, level2) \wedge WithinOne(level2, level3) \wedge (level1 \neq level2) \wedge (level2 \neq level3) \wedge (level1 \neq level3)) \rightarrow \neg WithinOne(level1, level3))$
  - If three levels are different, the first within one of the second and the second within one of the third, then the first is not within one of the third.
- $\forall level1 \forall level2 \forall level3((WithinOne(level1, level2) \wedge \neg WithinOne(level2, level3) \wedge (level1 \neq level2) \wedge (level2 \neq level3) \wedge (level1 \neq level3) \wedge LessThan(level1, level2) \wedge LessThan(Level2, Level3)) \rightarrow \neg WithinOne(level1, level3))$
  - If three levels are different and in ascending order, the first is within one of the second and the second is not within one of the third, then the first is not within one of the third.
- $WithinOne(a, b) \wedge WithinOne(b, c) \wedge WithinOne(c, d) \wedge WithinOne(d, e) \wedge WithinOne(e, f) \wedge WithinOne(f, g)$
  - Establish ranking of level constants within our domain.

- $\forall person1 \forall level1(MaxLevel(person1, level1) \leftrightarrow (Level(person1, level1) \land \neg(\exists person2 \exists level2(Level(person2, level2) \land LessThan(level1, level2)))))$
  - For every person, they have the max level if and only if there are no other players with a higher level.
- $\forall person1(Solo(person1) \leftrightarrow \exists level1(MaxLevel(person1, level1) \land \neg(\exists person2 \exists level2(Level(person2, level2) \land WithinOne(level1, level2) \land (person1 \neq person2)))))$
  - A person has a solo if and only if they have the max level and there are no other players within one level.
- $Tutti(low) \leftrightarrow \forall person(\exists level(Level(person, level) \land LessThan(level, f) \land LessThan(a, level)) \land \neg Solo(person))$
  - There is a low level tutti if and only if every player is greater than level zero, less than level five, and no player has a solo.
- $Tutti(high) \leftrightarrow \forall person \exists level(Level(person, level) \land LessThan(e, level))$
  - There is a high level tutti if and only if every player has a level greater than four.
- $End \leftrightarrow \forall person(Level(person, a))$
  - It is the end if and only if all players are at level zero.
- $Solo(alice) \leftrightarrow State(a)$
- $Solo(bob) \leftrightarrow State(b)$
- $Solo(charlie) \leftrightarrow State(c)$
- $Tutti(low) \leftrightarrow State(d)$
- $Tutti(high) \leftrightarrow State(e)$
- $End \leftrightarrow State(f)$
  - Establish state mappings.
- $State(g) \leftrightarrow (\forall level(LessThan(level, g) \land \neg State(level)) \lor \exists level1 \exists level2(State(level1) \land State(level2) \land (level1 \neq level2)))$
  - The uncertain state is reached if and only if no states are provable or more than one state is provable.

```
(defun decls ()
  (declare-sort 'person)
  (declare-sort 'level)
  (declare-constant 'a :sort 'level)
  (declare-constant 'b :sort 'level)
  (declare-constant 'c :sort 'level)
  (declare-constant 'd :sort 'level)
  (declare-constant 'e :sort 'level)
  (declare-constant 'f :sort 'level)
  (declare-constant 'g :sort 'level)

  (declare-constant 'alice :sort 'person)
  (declare-constant 'bob :sort 'person)
  (declare-constant 'charlie :sort 'person)

  (declare-relation 'LessThan 2 :sort '(level level))
  (declare-relation 'Lvl 2 :sort '(person level))
  (declare-relation 'MaxLvl 2 :sort '(person level))
  (declare-relation 'Solo 1 :sort '(person))
  (declare-relation 'WithinOne 2 :sort '(level level))
  (declare-relation 'FinishedSolo 1 :sort '(person)))

(defparameter *P1*
  '(forall ((?level :sort level)) (not (LessThan ?level ?level))) "Premise 1")

(defparameter *P2*
  '(forall
    ((?level1 :sort level) (?level2 :sort level))
    (implies (LessThan ?level1 ?level2) (not (LessThan ?level2 ?level1)))) "Premise 2")

(defparameter *P5*
 '(and (Lvl alice a) (Lvl bob d) (Lvl charlie b) (not (FinishedSolo bob))) "Premise 5")

(defparameter *P6*
 '(and (LessThan a b) (LessThan b c) (LessThan c d) (LessThan d e) (LessThan e f) (LessThan

(defparameter *P7*
  '(forall ((?level1 :sort level) (?level2 :sort level) (?level3 :sort level))
    (implies (and (LessThan ?level1 ?level2) (LessThan ?level2 ?level3)) (LessThan ?level1 ?
```

Figure 2.1: A screen capture of the SNARK code used to prove that Player B has a solo.

The final rule is determined by the player levels at runtime, and converted into a logical representation. For example:

- $Level(alice, b) \wedge Level(bob, e) \wedge Level(charlie, c) \wedge$
  $\forall person \forall level(Level(person, level) \rightarrow ((person = alice \wedge level = b) \vee$
  $(person = bob \wedge level = e) \vee (person = charlie \wedge level = c)))$
  - This input indicates the respective input levels of each player, and ensures that they are the only possible combination in the logical relation Level.

This particular set of levels results in the logical outcome that the ensemble state is Solo(bob), or State(b). Figure 2.1 on page 9 illustrates the formatting of constant, relation, and premise declarations for the proof of Player Bob having a solo. By requesting a resolution proof from SNARK (to be explained shortly) at a coordinate that doesn't have a proper recommendation, we may use the results of the proof as the proper recommendation ongoing. To validate this claim, we must understand how SNARK works.

SNARK proofs begin with a set of declarations including the objects within our domain, and the arguments we are given for the proof; this populates our knowledge base. When given a query, SNARK starts by converting the knowledge base arguments into Conjunctive Normal Form. In first—order logic, this means that each sentence is reduced to a conjunction of disjunction of atomic statements. The steps to converting a first—order logic statement to Conjunctive Normal Form are as follows (Russell and Norvig, 2003):

- Eliminate Implications
  - Reduce implications ($\rightarrow$) to their disjunctive form. ($A \rightarrow B \Rightarrow \neg A \vee B$)
- Move $\neg$ Inwards
  - Convert instances of $\neg\exists$ and $\neg\forall$ to $\forall\neg$ and $\exists\neg$
- Standardize Variables
  - If a variable is re—used in a quantifier in a different scope, the variable is re—named to avoid conflict. e.g. $\forall x(\exists y Likes(x, y) \wedge \exists y Likes(y, x))$ becomes $\forall x(\exists y Likes(x, y) \wedge \exists z Likes(z, x))$
- Skolemize
  - Skolemization is the process of removing existential quantifiers by elimina-

tion. This is similar to instantiating a variable in a quantifier to a particular constant. In our prior example, $\forall x(\exists y Likes(x, y) \land \exists z Likes(z, x))$ might become $\forall x Likes(x, B) \land Likes(C, x)$ However, this loses the dependency of variables y and z upon x, and changes the meaning entirely. Therefore, instead of replacing variables with constants, we replace them with Skolem functions — $\forall x Likes(x, F(x)) \land Likes(G(x), x)$

- Drop Universal Quantifiers
  - All remaining variables must be universally quantified, so universal quantifiers may be dropped. $Likes(x, F(x) \land Likes(G(x), x)$
- Distribute $\land$ Over $\lor$
  - This step flattens the remaining sentence into CNF by the distribution property of predicate logic.

At this point, SNARK uses a combination of memoization, the resolution inference rule, and paramodulation to prove (or disprove) the goal queried of the knowledge base. The resolution inference rule takes in two clauses and produces a resolvent clause, which is a conjunction over resolved terms. In first—order logic, a term in a clause is complementary (and requires resolution) to a term in the second clause if one unifies with the negation of the other. For example, $Animal(F(x)) \lor Loves(G(x), x)$ and $\neg Loves(u, v) \lor \neg Kills(u, v)$ contain complementary literals $Loves(G(x), x)$ and $\neg Loves(u, v)$. They may be eliminated by replacing occurrences of u with G(x) and occurrences of v with x, resulting in $Animal(F(x)) \lor \neg Kills(G(x), x)$. Paramodulation essentially handles rules of variable equality and substitution. For instance, if $x = y$ and we resolve on (x, z), then $z = y$ and anywhere z appears, y may be substituted and vice versa.

As a brief example of how this works, we examine the Dreadsbury Mansion Mystery (Bringsjord, Bringsjord).

> Someone who lives in Dreadsbury Mansion killed Aunt Agatha. Agatha, the butler, and Charles live in Dreadsbury Mansion, and are the only people who live therein. A killer always hates his victim, and is never richer than his victim. Charles hates no one that Aunt Agatha hates. Agatha hates everyone except the butler. The butler hates everyone not

richer than Aunt Agatha. The butler hates everyone Agatha hates. No one hates everyone. Agatha is not the butler.

The formalization of the standard interpretation is as follows:

- $\forall person(LivesMansion(person) \leftrightarrow (person = agatha \vee person = butler \vee person = charles))$
  - Agatha, the butler, and Charles are the only people who live in the mansion.
- $\exists person(Killed(person, agatha) \wedge LivesMansion(person))$
  - There is a person who killed Agatha and lives in the mansion.
- $\forall person1 \forall person2(Killed(person1, person2) \rightarrow (Hates(person1, person2) \wedge \neg Richer(person1, person2)))$
  - If a person kills someone else, they hate the person they killed and are not richer than them.
- $\forall person(Hates(agatha, person) \rightarrow \neg Hates(charles, person))$
  - If Agatha hates someone, Charles does not hate them.
- $\forall person((person \neq butler) \leftrightarrow Hates(agatha, person))$
  - Agatha hates someone if and only if they are not the butler.
- $\forall person(\neg Richer(person, agatha) \rightarrow Hates(butler, person))$
  - The butler hates anyone not richer than Agatha.
- $\forall person(Hates(agatha, person) \rightarrow Hates(butler, person))$
  - Everyone agatha hates, the butler hates.
- $\forall person1(LivesMansion(person) \rightarrow \exists person2(LivesMansion(person2) \wedge \neg Hates(person1, person2)))$
  - No one in the mansion hates everyone in the mansion.
- $agatha \neq butler$

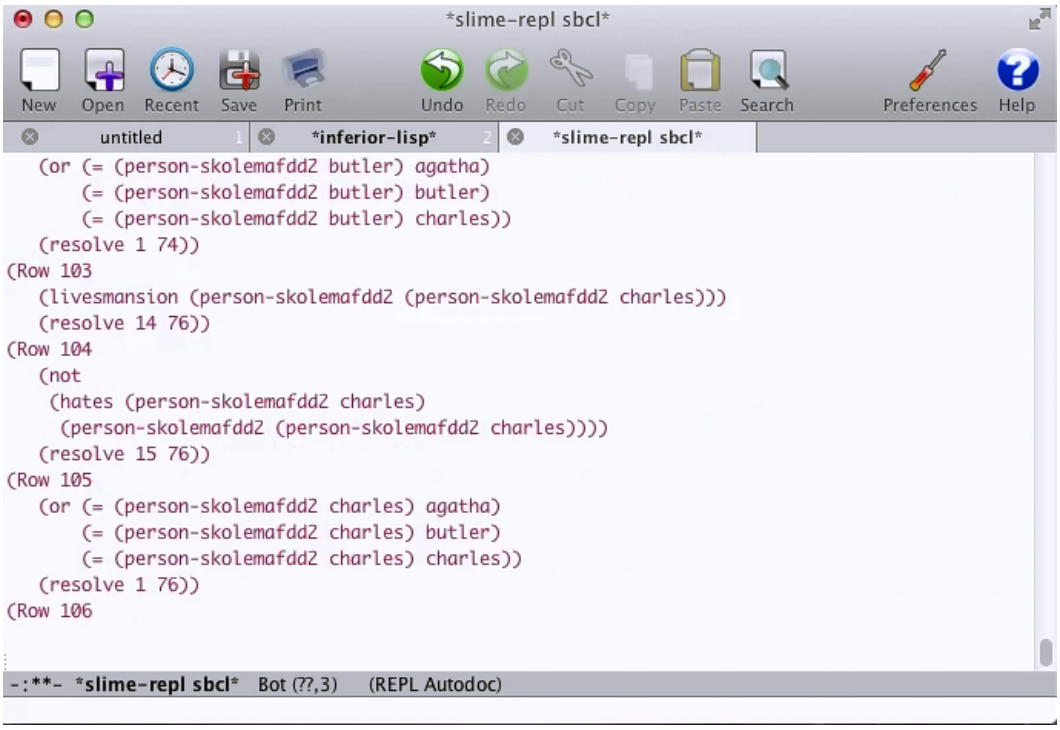

Figure 2.2: A screenshot of the SNARK resolution proof for the Dreadsbury Mansion problem.

We give a very informal proof illustrating that Agatha killed herself and present a SNARK resolution proof of the same problem (Figure 2.2 above):

- There exists someone who killed Agatha and lives in the Mansion.
- Agatha, the Butler, and Charles live in the mansion, so one of them must be the killer.
- The butler did not kill Agatha:
  - He hates everyone Agatha hates. (Agatha, Charles)
  - He does not hate himself, because no one hates everyone.
  - He must be richer than Agatha, otherwise he would hate himself.
  - He is richer than Agatha, so he did not kill her.
- Charles did not kill Agatha:
  - Charles does not hate anyone Agatha hates.
  - Agatha hates everyone who is not the Butler. (Agatha, Charles)
  - Charles does not hate Agatha, so he did not kill her.

- The only remaining person must have killed Agatha.

Returning to the original problem, we establish a knowledge base with each of the original premises given, and then apply the musician parameters when querying to confirm a particular state. Since we do not yet know which state the player is in, we begin by attempting to prove the Uncertainty state. However, if there exactly one state that is true (and not the uncertainty state), we then query the knowledge base for the remaining states. Since the proof for the uncertain state checks every other state, the query involves no further computation.

3. MULTIPLE—VARIABLE RECOMMENDATIONS

3.1 Representation and Heuristic

As observed by Simon Ellis (Ph.D. Student at RPI, researcher on CAIRA project), the axes of a parameter recommendation cube correspond to each players input and there are three inputs, so we may visually represent each state as a color. The working interpretation is that a color corresponds to the most extreme location for a state (see Figure 3.1 on page 16). For instance, the most extreme state in which player A has a solo would have a specific parameter set at 6, 0, 0, corresponding to red. Solo B is 0, 6, 0 which encodes to green, and Solo C is 0, 0, 6 — blue. The color for an uncertain state is given by the coordinates it lies at.

It is my hypothesis that by converting states to colors in this fashion, we may perform chromatic averaging across parameters to obtain a best guess for all of them. That is, for each variable we look up the state as previously calculated in the single—variable recommendation cube, and average the RGB values that correspond to each state. Furthermore, this allows for different parameters to be weighted more strongly than others. We may take the result of the chromatic average, and look up the state at the corresponding coordinates. If one parameter indicates Solo A (6, 0, 0 — red — 1, 0, 0), and another indicates Solo B (0, 6, 0 — green — 0, 1, 0), the resulting dark yellow (.5, .5, 0) falls in an uncertain state (3, 3, 0). The goal is to compare the recommended state via this chromatic averaging heuristic to the recommendation generated by a proof utilizing multiple parameters.

3.2 Multiple—Variable Definitions

Similar to the single—variable proofs and definitions, we must understand the six possible variables as well as their relations to each other. The six variables are as follows: two types of tension ($\texttt{Tension}_B$ and $\texttt{Tension}_{MIR}$), `Dynamic`, `Tempo`, `Valence` and `Activity`. Tension refers to "inner striving, unrest, or imbalance often with physiological indication of emotion (Merriam-Webster, c)." Applied to music, higher levels of activity, energy, volume and roughness indicate high levels

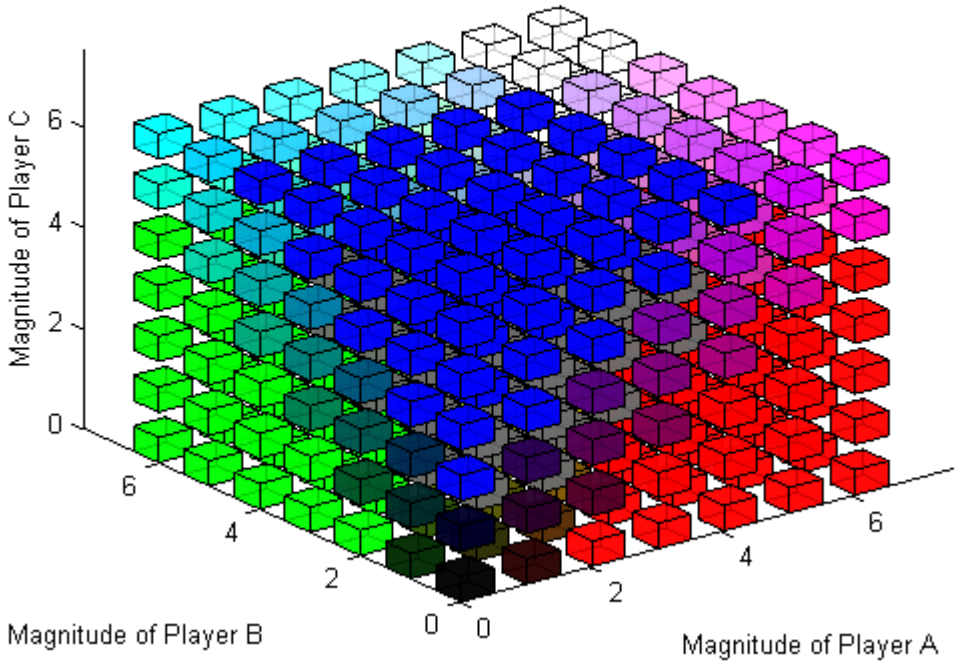

Figure 3.1: The chromatic representation of a single—feature recommendation cube. The cube was generated by performing recommendation proofs at each set of coordinates. We see that uncertain states occur around the boundaries of the tutti states.

of tension. $\texttt{Tension}_B$ refers to the variable defined by Braasch et al. (2011), and factors in roughness (note density) and loudness. $\texttt{Tension}_{MIR}$ refers to the variable defined by the Music Information Retrieval (MIR) Toolbox and is calculated from the standard deviation of the root-mean-square across frames (indicative of energy roughness), the Harmonic Change Detection Function (tonal chaos), key uncertainty, and novelty (Lartillot, 2011; Eerola et al., 2009). `Dynamic` is the measurement of loudness for a period of time (Lartillot, 2011). `Activity` is a measure of average energy across frames as well as brightness of the sound (Lartillot, 2011; Eerola et al., 2009). Tempo is "the rate of speed of a musical piece or passage (Merriam-Webster, b)," and is calculated from the note onset curve (Lartillot, 2011). Finally, `Valence` is the measure of emotion type: a low `Valence` indicates emotions like anger, sadness

or fear (depending upon activity), while a high `Valence` corresponds to happiness or tenderness (Lartillot, 2011; Eerola et al., 2009). It is calculated by many of the same parameters as $\texttt{Tension}_{MIR}$, though some correlations are positive instead of negative, and we see the effects of this later.

3.3 Multiple Variable State Determination

Now that we have definitions and determined the origins of each parameter, we must construct a set of rules for each unique combination. We start by examining the relations between variables. Although $\texttt{Tension}_B$ and $\texttt{Tension}_{MIR}$ both factor in roughness, $\texttt{Tension}_B$ is primarily based on loudness and $\texttt{Tension}_{MIR}$ focuses primarily on tonality and chaoticness; therefore, these two must be treated separately when constructing a relation. `Dynamic` is heavily related to $\texttt{Tension}_B$ such that any inference based upon loudness would be nearly equivalent to a conclusion drawn using $\texttt{Tension}_B$. Given how activity is calculated, it is possible for there to be high activity and low tension/loudness, or vice versa. However, this may be used as a differentiating factor during a low tension tutti. Recalling that an improvisational solo occurs when one player's improvisation is played over the others, and that the others are playing in an unobtrusive manner to the solo, a low tension tutti in which one player has significantly more activity than the others may be considered a solo.

`Tempo` is partially related to $\texttt{Tension}_{MIR}$, as high tempo indicates a densely populated note onset curve, which also indicates high chaos and therefore tension. It mostly unrelated to $\texttt{Tension}_B$, as a loud, drawn-out note can potentially be high tension. Since there is a single note onset, however, tempo is calculated to be slower. With regards to `Valence`, the calculation is similar to $\texttt{Tension}_{MIR}$ but more positively correlated so there is an inverse relationship between the two variables. High tension portions of a song will tend to be low `Valence`, as low tension elements will tend to be high `Valence`. However, as `Valence` is a measure of emotion during a section of audio, there are no inferences we can make that tie it specifically to the state of the music, as any state may be any emotion. Inferences on `Valence` may be used at a later point to direct CAIRA how to play a particular state.

Again referring to the rule that the other players are to be unobtrusive during

a solo, we may begin to hone in on the most important aspects to base the relations upon. A solo involves exactly one player that stands out from the others, a tutti is where all players are perceptibly playing together, and the end occurs when no one is playing. Given this, the easiest way to stand out from other players is first by `Dynamic` or $\texttt{Tension}_B$, and then next by $\texttt{Tension}_{MIR}$, `Activity`, and `Tempo`. So, if a player is significantly louder than the others, then it is most likely a solo regardless of content. If all players have a similar dynamic, then we look to the other variables to determine whether or not a player stands out from the others. Using the single—variable proof as a basis (as well as LessThan, WithinOne, and the State operators), we may roughly formalize the previous notion as follows:

- $\forall person1 \forall level1(MaxDyn(person1, level1) \leftrightarrow (Dyn(person1, level1) \wedge \neg(\exists person2 \exists level2(Dyn(person2, level2) \wedge LessThan(level1, level2)))))$
  - There is a person with a max `Dynamic` if and only if there are no players that have a higher dynamic.
- $\forall person1 \forall level1(MaxTenB(person1, level1) \leftrightarrow (TenB(person1, level1) \wedge \neg(\exists person2 \exists level2(TenB(person2, level2) \wedge LessThan(level1, level2)))))$
  - There is a person with max $\texttt{Tension}_B$ if and only if there are no players with a higher tension.
- $\forall person1 \exists level1(Solo(dyn, person1) \leftrightarrow (MaxDyn(person1, level1) \wedge \neg(\exists person2 \exists level2(Dyn(person2, level2) \wedge (person1 \neq person2) \wedge WithinOne(level1, level2)))))$
- $\forall person1 \exists level1(Solo(TenB, person1) \leftrightarrow (MaxTenB(person1, level1) \wedge \neg(\exists person2 \exists level2(TenB(person2, level2) \wedge (person1 \neq person2) \wedge WithinOne(level1, level2)))))$
- $\forall person((Solo(dyn, person) \vee Solo(TenB, person)) \rightarrow Solo(person))$
  - The preceeding three rules establish that if there is someone with the highest `Dynamic` or highest $\texttt{Tension}_B$ then they have a solo based on that parameter. If either parameter indicates a solo for that player, then they have a solo.
- $Tutti(dyn, low) \leftrightarrow \forall person \forall level(Dyn(person, level) \wedge LessThan(level, f) \wedge LessThan(a, level) \wedge \neg Solo(dyn, person))$

- $Tutti(TenB, low) \leftrightarrow \forall person \exists level(TenB(person, level) \wedge LessThan(level, f) \wedge LessThan(a, level) \wedge \neg Solo(TenB, person))$
- $Tutti(dyn, high) \leftrightarrow \forall person \exists level(Dyn(person, level) \wedge LessThan(e, level))$
- $Tutti(TenB, high) \leftrightarrow \forall person \exists level(TenB(person, level) \wedge LessThan(e, level))$
  - The preceeding four rules establish the Tutti states for $\texttt{Tension}_B$ and `Dynamic`.
- $\forall person1 \forall level1(MaxTenMIR(person1, level1) \leftrightarrow (TenMIR(person1, level1) \wedge \neg(\exists person2 \exists level2(TenMIR(person2, level2) \wedge LessThan(level1, level2)))))$
  - There is a person with max $\texttt{Tension}_{MIR}$ if and only if there are no players that have a higher tension.
- $\forall person1 \forall level1(MaxAct(person1, level1) \leftrightarrow (Act(person1, level1) \wedge \neg(\exists person2 \exists level2(Act(person2, level2) \wedge LessThan(level1, level2)))))$
  - There is a person with max `Activity` if and only if there are no players that have a higher `Activity`.
- $\forall person1 \forall level1(MaxTemp(person1, level1) \leftrightarrow (Temp(person1, level1) \wedge \neg(\exists person2 \exists level2(Temp(person2, level2) \wedge LessThan(level1, level2)))))$
  - There is a person with max `Tempo` if and only if there are no players that have a higher `Tempo`.
- $(\forall person \neg(Solo(dyn, person) \vee Solo(TenB, person)) \wedge (Tutti(dyn, low) \vee Tutti(dyn, high) \vee Tutti(TenB, low) \vee Tutti(TenB, high))) \rightarrow \forall person1(((Solo(TenMIR, person1) \vee Solo(activity, person1) \vee Solo(tempo, person1)) \wedge \neg \exists person2(Solo(TenMIR, person2) \vee Solo(activity, person2) \vee Solo(tempo, person2))) \rightarrow Solo(person1))$
  - This rule states that if `Dynamic` and $\texttt{Tension}_B$ result in Tutti recommendations, check against the other parameters. If exactly one person has a solo recommendation from the other parameters, then it is a solo. Otherwise, it remains a tutti.

- $\forall person \neg Solo(person) \rightarrow ((Tutti(dyn, high) \vee Tutti(TenB, high)) \rightarrow Tutti(high))$
  - If there are no solos, then if `Dynamic` or $\texttt{Tension}_B$ are high level tutti, the state is a high level tutti.
- $(Tutti(dyn, low) \wedge Tutti(TenB, low)) \rightarrow Tutti(low)$
  - If both dynamic and Tension_B are low level tutti, then there is a low level tutti.
- $\forall person(Dyn(person, a) \wedge TenB(person, a) \wedge TenMIR(person, a) \wedge Tempo(person, a) \wedge Activity(person, a)) \leftrightarrow End$
  - If every person has level zero for every variable, then it is the end.

4. RESULTS

4.1 Multiple Variable State Cube

We may generate a state cube from these rules in a similar manner to the single—variable proofs. However, as there are five variables, we must flatten them along each player dimension. One such method is to index on a particular variable(s), and then create a linear equation for lookup. For example, we can index in the order of $\texttt{Tension}_B$, `Dynamic`, $\texttt{Tension}_{MIR}$, `Activity`, `Tempo` where the A, B, and C coordinates are determined as follows:

$$A = 7^4 * A_{Tension_B} + 7^3 * A_{dynamic} + 7^2 * A_{Tension_{MIR}} + 7 * A_{activity} + A_{tempo}$$
$$B = 7^4 * B_{Tension_B} + 7^3 * B_{dynamic} + 7^2 * B_{Tension_{MIR}} + 7 * B_{activity} + B_{tempo}$$
$$C = 7^4 * C_{Tension_B} + 7^3 * C_{dynamic} + 7^2 * C_{Tension_{MIR}} + 7 * C_{activity} + C_{tempo}$$

The state cube for two variables is in Appendix A (Figure A on page 27). It is hard to illustrate much more than a two-variable cube, as the number of states is 7^{n*3} over n variables. We may still determine the error of a cube from its logical recommendation by calculating the state each index individually. However, we must also note that although this logic covers every combination of input, certain inputs cannot occur because they are related to one another. For instance, it is impossible to have 0, 0, 0 loudness and 6, 6, 6 Tension_B (this being an extreme example).

By iterating through all possible states, and comparing the result of the chromatic averaging heuristic to the proof results, we can determine a percentage of correct states. However, this operation occurs in $O(7^n)$ time where n is the number of variables we are using. Furthermore, for n less than five, there are two major unique ways to combine the variable types. This is because $\texttt{Tension}_B$ and `Dynamic` are on one level of contribution, and the remaining variables are on another. At least one of the former is necessary for determining state, so for $n = 2$ through $n = 4$ there is a unique combination where there is one "primary" variable and one where there are two "primary" variables. The results of the comparison for $n = 2$ through $n = 4$ are available in Figure 4.1 on page 22. It is noted that to calculate all states on five variables, it would take an estimated 217 days computation time

Table 4.1: The results of the comparison on each unique primary/secondary variable combination for n = 2 through n = 4.

Variable Combination	States Correct	Total States	Percentage Correct
Two Vars, Two Primary	53448	117649	45.430
Two Vars, One Primary	56399	117649	47.938
Three Vars, Two Primary	9037005	40353607	22.395
Three Vars, One Primary	16690235	40353607	41.360
Four Vars, Two Primary	2587977821	13841287201	18.698
Four Vars, One Primary	4263168056	13841287201	30.800

(runtime on four variables in seconds multiplied by 343).

The results show that as the number of variables (and also states) increase, the accuracy of the proposed heuristic decreases. Furthermore, we note that for no variable level is the heuristic within a 95 percent level of certainty. What we have gained from this an interesting method of visually representing scalar data. We also gain a glimpse into encoding aural data as visual (synesthesia) and may possibly be used in reverse to some degree. That is to say, taking visual formations and translating them to state data. However, the recommendation heuristic that utilizes this representation needs much improvement before it is considered accurate enough to replace the nearest—neighbor search.

5. NEXT STEPS

While the heuristic is unusable as is, there are a number of additional approaches to explore that may increase accuracy. One such method is altering the weights for each of the input variable state—chromas before averaging. This would allow us to place emphasis on a particular tier of variable, and be more reflective of a state recommendation should the user wish to alter the importance of a variable over another (a future improvement planned for the CAIRA project).

Another approach is to change the method of state—chroma combination. There are several methods which exist to add colors together. The averaging method used for this thesis is useful for creating gradients and lightening or darkening colors. One other approach is to add the colors together, and take the lesser of the sum or the maximum color value. This creates lighter colors, but may also result in unintended effects. One may also multiply RGB values together and rescale the magnitude by a power of 255, producing darker colors.

It is also possible that the heuristic presented within the thesis does work within the set of mathematically possible parameters. As mentioned in the Results discussion, there are certain configurations of parameters which are simply not possible due to dependencies on similar features. For the feature combinations that are possible, it is possible that the heuristic is correct and therefore usable. This requires mathematical analysis beyond the author's knowledge.

We may also try discarding discretized state colorization as we have at present, and dealing with gradients. Because we are forcing a wide range of state data to adhere to one color, it is possible that we are losing out on some precision that may help with accuracy. In other words, rather than explicitly classifying a solo as red, green or blue, tutti as grey, etc., we allow each coordinate to represent its own color. The more distinct a color, the more certain we are about the recommendation. Darker colors indicate less energy, and bright colors correspond to more energy. Granted, this adds some degree of complexity when definitively map one color to a state, but it may add large amounts of accuracy.

One subject to contemplate regarding the heuristic (and other algorithms based upon the state data structures) is that a linear search is technically NP—complete when the input is expressed in terms of number of variables instead of individual states. It is a future goal to either make the data structure more efficient (if possible), or determine a method of verifying assignment in polynomial time.

Branching further, it is also possible to take player inputs and train a machine learning algorithm to approximate state recommendations. Since neural networks may be represented as matrices of weights, a state computation takes polynomial time via matrix multiplication. If there is a missing state, the network may be used for state estimation, and the result of the logical proof can be back—propagated to reduce error (Russell and Norvig, 2003). Similar to the nearest—neighbor approach, we expect estimated states to become more accurate. However, this is because a network corrects itself for error, whereas the nearest—neighbor approach relies on the increase of state data within the matrix (and therefore closer neighbors). A neural network may also take into account player tendencies and adjust recommendations accordingly. For instance, it is possible for a player to produce high tension curves as a stylistic trait, but this might normally be misinterpreted as a solo. An intriguing observation about this approach is that it is a hybridization of the two major branches of Artificial Intelligence.

Finally (and most importantly), the logical premises for state recommendations are very rough and may require refinement. Furthermore, it is also the intent to integrate these state proofs with a more cognitively robust model which factors in more of the music structure (e.g. note, duration, pitch, timbre, harmonics, etc.). As the goal of the CAIRA project is to understand and develop creative agents, a joining between music state and music structure would be very useful for music composition and direction as well as future models of machine understanding.

BIBLIOGRAPHY

Braasch, J. (2012, September). The μ-cos$m$ project: an introspective platform to study intelligent agents in the context of music ensemble improvisation. (to appear).

Braasch, J., S. Bringsjord, C. Kuebler, P. Oliveros, A. Parks, and D. V. Nort (2011, 10). Caira – A Creative Artificially–Intuitive and Reasoning Agent as conductor of telematic music improvisations. In *Audio Engineering Society Convention Paper*.

Bringsjord, S. The Dreadsbury Mansion Mystery. http://homepages.rpi.edu/~brings/LOGARG/dread/dread.html. (Date Last Accessed April, 16, 2013).

Brotman, D. A Passion For Jazz. http://www.apassion4jazz.net/glossary4.html. (Date Last Accessed April, 16, 2013).

Cope, D. (2000). *The Algorithmic Composer*. Madison: AR-Editions, Inc.

Cope, D. (2001). *Virtual Music: Computer Synthesis of Musical Style*. Cambridge: MIT Press.

Cope, D. (2006). *Computer Models of Musical Creativity*. Cambridge: MIT Press.

Cope, D. (2008). *Hidden Structure: Music Analysis Using Computers*. Madison: AR-Editions, Inc.

Eerola, T., O. Lartillot, and P. Toiviainen (2009). Prediction of Multidimensional Emotional Ratings in Music From Audio Using Multivariate Regression Models. In *International Conference on Music Information Retrieval*.

Ellis, S., N. S. Govindarajulu, S. Bringsjord, A. Haig, C. Kuebler, J. Taylor, J. Braasch, P. Oliveros, and D. V. Nort (2012). Creativity and Conducting: Handle in the CAIRA Project. In *Proceedings of the Workshop "Computational Creativity, Concept Invention, and General Intelligence"*.

Lartillot, O. (2011, January). *MIRtoolbox 1.3.2: User's Manual*. University of Jyväskylä, Finland: Finnish Centre of Excellence in Interdisciplinary Music Research.

Merriam-Webster. Solo - Definition and More from the Free Merriam-Webster Dictionary. http://www.merriam-webster.com/dictionary/solo. (Date Last Accessed April, 16, 2013).

Merriam-Webster. Tempo - Definition and More from the Free Merriam-Webster Dictionary. http://www.merriam-webster.com/dictionary/tempo. (Date Last Accessed April, 16, 2013).

Merriam-Webster. Tension - Definition and More from the Free Merriam-Webster Dictionary. http://www.merriam-webster.com/dictionary/tension. (Date Last Accessed April, 16, 2013).

Merriam-Webster. Tutti - Definition and More from the Free Merriam-Webster Dictionary. http://www.merriam-webster.com/dictionary/tutti. (Date Last Accessed April, 16, 2013).

Russell, S. and P. Norvig (2003). *Artificial Intelligence: A Modern Approach* (2nd ed.). Upper Saddle River: Pearson Education, Inc.

Stickel, M. E. SNARK - SRI's New Automated Reasoning Kit. http://www.ai.sri.com/~stickel/snark.html. (Date Last Accessed April, 16, 2013).

APPENDIX A
ADDITIONAL STATE PLOTS

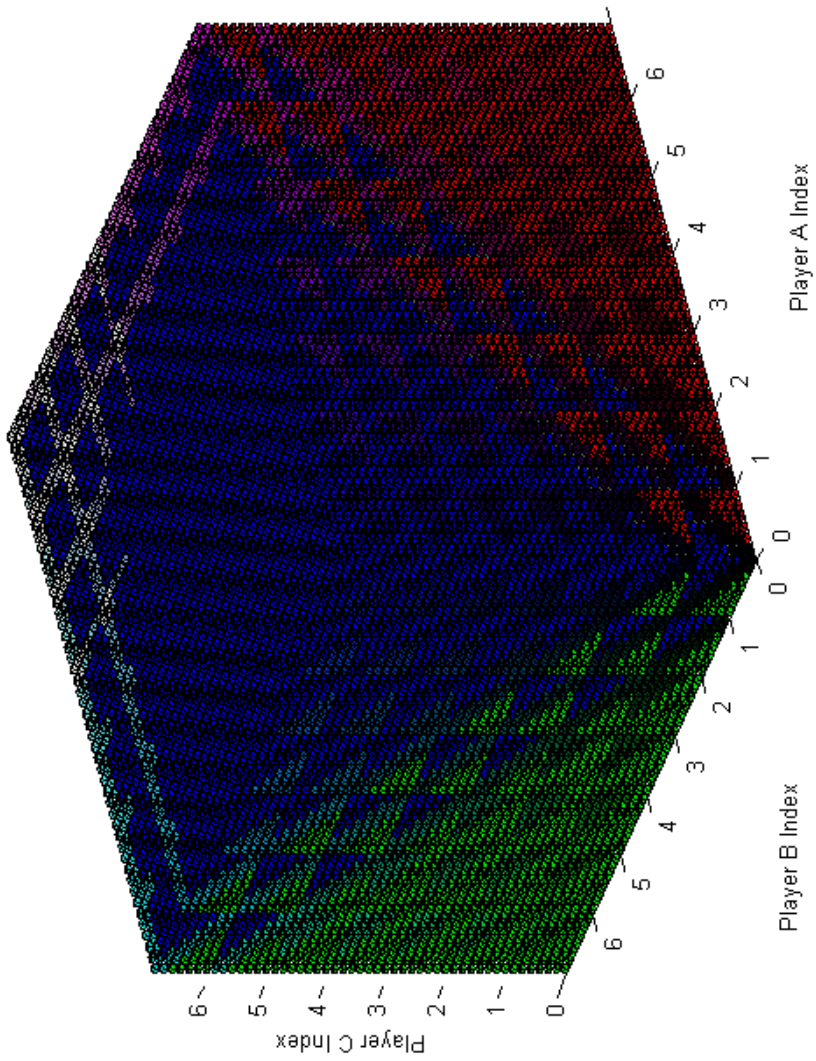

Figure A.1: The chromatic cube representing the addition of two single state variables. Given the algorithm to construct it, there is observable fractal—like behavior.