COMPLEX SYSTEMS DATA MINING AND KNOWLEDGE EXTRACTION

by

DAVID BRANDON LUPER

(Under the Direction of HAMID R. ARABNIA)

ABSTRACT

Complex systems appear across numerous disciplines, and analyzing them can be difficult. Standard analysis techniques fail to capture concepts such as emergent behavior, self-organization, or the entanglement among related components within a system. A better knowledge of complex systems could help avoid financial system collapse, predict terrorist network actions, and fight disease.

One way to understand a complex system better is to leverage the information encapsulated within the higher order relationships of the system. A complex system is a set of interconnected compartments, and it is these connections that give rise to the characteristics and complexity of the system. These relationships define the structure of a network and the flow across them defines the function. The structure and function of a system encodes valuable information about the system, information that can be hard to find due to the massive amount of information contained within a complex system. In order to isolate important information, data analysis techniques must be implemented. The field of data mining is perfectly suited for this task. Data mining is a term used to describe a compilation of techniques including statistics, artificial intelligence, computational intelligence and database management used to discover and extract information in an automated fashion from large data sets. Though not universal, many

forms of data mining are restricted to numerical input. This can be problematic when analyzing a system modeled as a graph, which is of a symbolic nature. Another problem with complex system analysis is a disconnect between higher level system function and lower level compositional elements within the system. The work herein proposes a methodology to solve these problems by presenting an encoding framework to map a complex system of connected symbols into a meaningful numeric feature space. This methodology will allow numerous techniques from the field of data mining to be applied in transformative ways, creating new possibilities in the field of systems research.

INDEX WORDS: Data Mining, Knowledge Extraction, Complex Systems Analysis, Symbolic Sequence Mining, Symbolic Time Series, Encoding Framework, Network Flux Analysis, Pattern Discovery

COMPLEX SYSTEMS DATA MINING AND KNOWLEDGE EXTRACTION

by

DAVID BRANDON LUPER

B.S., Emmanuel College, 2004

A Dissertation Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

ATHENS, GEORGIA

2012

COMPLEX SYSTEMS DATA MINING AND KNOWLEDGE EXTRACTION

by

DAVID BRANDON LUPER

Major Professor:	Hamid R. Arabnia
Committee:	John Schramski Walter D. Potter Khaled Rasheed

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2012

DEDICATION

This work is dedicated to my mother Janice Renae Luper who lovingly cared for me in my youth and had to depart this world early. Your immense love shaped me early in my life, and I would not be where I am without you. You are missed so much and loved beyond measure.

ACKNOWLEDGEMENTS

This work would not have been possible without the help and groundwork laid by Caner Kazanci; in addition, the assistance of the System Ecology and Engineering research group and VIGRE research group helped shape the work presented herein. Recognition also needs to be paid to Dera Weaver and Iniray Luper who helped significantly towards making the literary composition of this work better.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

Complex systems appear across numerous disciplines, and analyzing such systems can be difficult. Standard analysis techniques fail to capture or understand the intricacies of concepts such as emergent behavior, self-organization, or the entanglement among related components within a system. Complex systems are used by numerous disciplines to model processes in the natural world, and a better knowledge of complex systems could help avoid financial system collapse, predict terrorist network actions, and fight disease.

One way to understand a complex system better is to leverage the information encapsulated within the relationships of the system. A complex system by definition is a set of interconnected compartments, and these connections give rise to the characteristics and complexity of the system. These relationships define the structure of a network, and the flow across them defines the function. The structure and function of a system holds valuable information about the system, encoded information that can be hard to obtain due to the massive amount of information contained within a complex system. In order to isolate important information, research must implement a variety of data analysis. The field of data mining seems perfectly suited for this task.

Data mining is a term used to describe a compilation of techniques including statistics, artificial intelligence, computational intelligence and database management used for discovering and extracting information in an automated fashion from large data sets. Many, though not all, forms of data mining are restricted to numerical input, and when analyzing graphs, which are

symbolic in nature, it can be difficult to use numeric analysis methods. The work herein proposes a methodology to solve this problem, allowing numerous techniques from the field of data mining to be applied in transformative ways. These transformations will present new possibilities in the field of systems research.

This dissertation is intended to address complex systems analysis through contextual investigation of calculated nodal groupings within a system. It is the idea herein that relationships uniquely define a system, and through these relationships a better understanding of complex systems can be attained. To date, most research into complex systems deals with analysis of individual node, ignoring the contextual information about each node's role in the system. The extent of effect a particular node can have on a system is potentially far reaching. To completely define relationships for a node in a graph, one must look past direct connections through single edges, to the set of relational groupings that define a node's indirect effects on the system. It is through these indirect relational connections that distant nodes within a graph can have far-reaching effects on their relational counterparts. This paradigm captures the entirety of the nodal context.

Defining nodal context in this way poses a problem to systems analysis, as there is exponential growth in the number of total nodal groupings within a system. Using techniques outlined herein this problem can be overcome to provide powerful analytic capabilities. With increased contextual nuance, a more complete picture of the system can be established for analysis; however, the complexity involved in these systems presents massive amounts of data to analyze. The current work uses a computational methodology to map a system into a numerical feature space so other machine learning and data mining techniques can be employed to perform analysis. The fields of machine learning and computational intelligence have garnered much

attention recently due to their effectiveness in allowing a computer to learn concepts defined by data distributions. Many techniques in these fields (*i.e.* Neural Networks, Self-Organizing Feature Maps, Fuzzy Logic and Support Vector Machines) rely on an algorithm's ability to divide a numeric feature space into sections that define a certain concepts about a data distribution. The numeric feature space computed by the methodology herein allows machine learning and computational intelligence to be used on systems analysis in ways not previously possible.

Due to the fact that complex systems appear in numerous disciplines, the effect of this methodology is potentially far reaching. If gains can be made in complex systems research, then any field where complex systems are present will benefit. The field of social theory suggests an example: Graphs of social interaction in this field represent complex systems, and better analysis techniques could yield increases in the ability to disrupt a network's ability to transmit information, provide better event detection based on system behavior and allow for an increased ability to detect hierarchal rankings of importance within a system. Another example can be taken from the field of bioinformatics: With better systems analysis techniques it could be possible to better isolate non coding RNA regions within chromosomes, better understand genetic causes of diseases and heighten the ability to distinguish different types of protein folding structures from DNA sequences. As a final example, the field of economics could greatly benefit from an increased understanding of complex systems, thus yielding better financial system vulnerability assessment, higher accuracy market predictions and more in depth market impact studies.

When complex systems present themselves in any field, it is common for researchers in that area to model the system. The proposed methodology performs computational techniques

on data simulated over models to present important information not attainable otherwise. The ubiquity of complex systems combined with generalized analysis techniques poses novel, high impact results.

The specific objectives of the current work will be to author computational tools to perform the following tasks:

1. Isolate relational groupings within a system (*i.e.* structurally decompose a complex system into a set of fundamental relationships)
2. Analyze functional activity of relational groupings within a system using quantifiable, numeric metrics
3. Leverage information contained in functional analysis towards data mining and knowledge extraction

Accomplishing these objectives will require a broad tool set from computational and mathematical fields. As this work suggests a new approach to complex systems analysis, formal definitions of new concepts will need to be ascertained en route to achieving the proposed objectives. In addition to addressing these objectives, this work will delve into application of the resulting methodologies in order to demonstrate their effectiveness.

Overview

There is currently a disconnect in the field of systems analysis between high order system behavior/traits and the lower level compositional elements of the system. It is known that systems exude behaviors and traits. It is also known that systems are comprised of smaller compositional elements. The massive amount of information in complex systems makes it difficult to analyze how compositional elements drive or encode these higher level functions

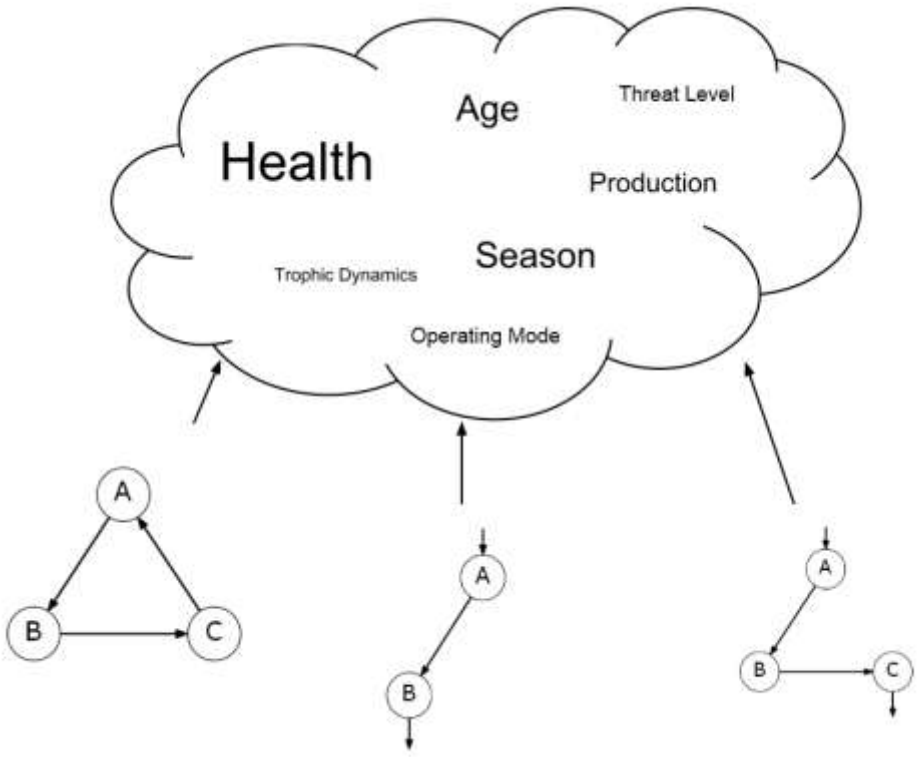


Figure 1: This figure shows basic compositional units of a system contributing to higher level system behavior/traits.

(Figure 1). This work seeks to bridge this gap by providing a means of quantifying how each of these compositional elements within a system contributes towards higher order function.

The first objective of this work, as stated earlier, is to isolate fundamental structural units within a graph. These structural units are the compositional elements contributing to higher level system function. Figure 2 shows a simple example network that will be used to illustrate the process of structural decomposition.

This simple network of three nodes will be used as an example network to illustrate the structural and functional decomposition objectives of this work. Structural decomposition is performed first and is the process of finding all the basic compositional elements in the system. This will be explained in detail later, but the general concept is to find all the simple cycles in the

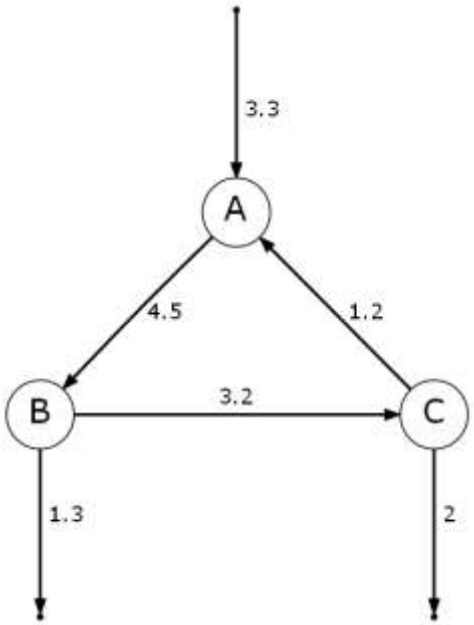


Figure 2: Example system.

graph. This work introduces a new domain-generic term for these simple cycle/compositional elements. They are referred to herein as fluxes. This example graph has inputs and outputs that are grouped together as going to and coming from an "environment" (a separate node/system). For the purposes of decomposition the environment is just another compartment in the adjacency matrix. It will be noted later that from a domain standpoint the environment can take on special meaning during the analysis of the coefficient vector produced by this methodology. A structural decomposition of the example network is shown in Figure 3.

The structural fluxes present a group of compositional elements whose activity can be analyzed in relation to high level system traits. The activity analyzed is how much flow from the system can be attributed solely to each compositional element. This is referred to herein as functional decomposition. A current method for accomplishing this functional decomposition is to repeatedly find an edge in the system with the least flow in the graph and remove a cycle containing that edge. That methodology works for this simple example and produces the

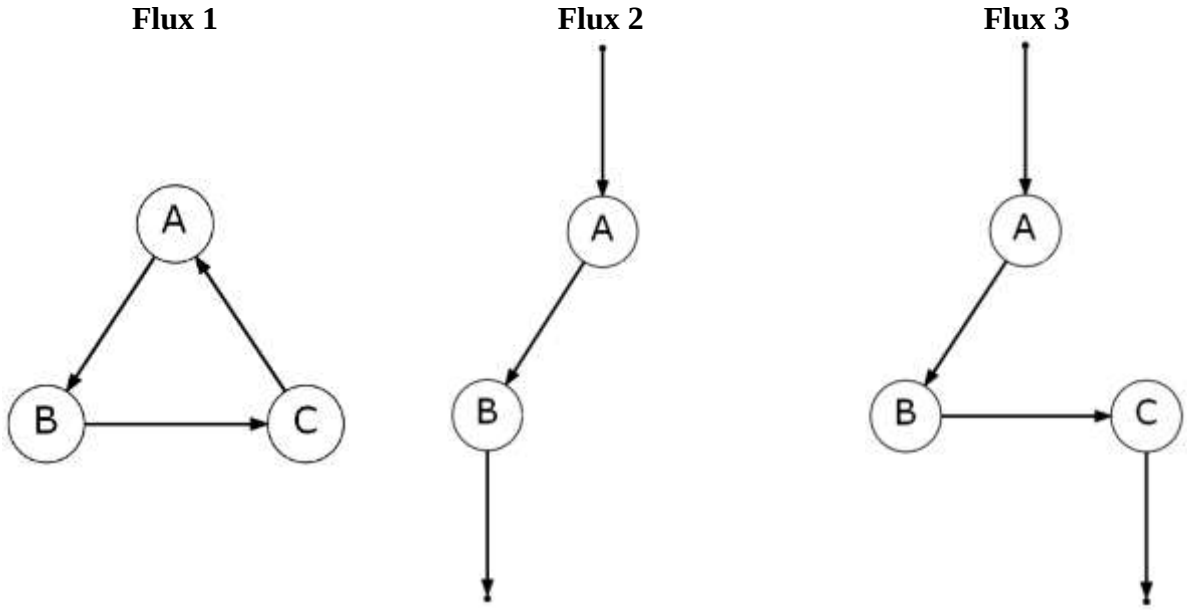


Figure 3: All fluxes in the example system from Figure 2.

functional decomposition shown in Figure 4 and Figure 5. The functional decomposition here would find the edge with the least flow A → C (1.2) and remove the cycle it belongs to, Flux 1. The smallest edge left after that would be B → Environment (1.3); it belongs to Flux 2. Finally, the only flux left in the system is Flux 3. Every time a minimum edge is found in this example the coefficient assigned to the removed flux containing that edge is the total flow for that edge.

Flux 1	Flux 2	Flux 3
1.2	1.3	2

Figure 4: This figure illustrates a functional decomposition in vector format. The vector contains an index for each of the structural fluxes in the system. The value at each of the indexes represents the amount of system flow belonging solely to each of the fluxes.

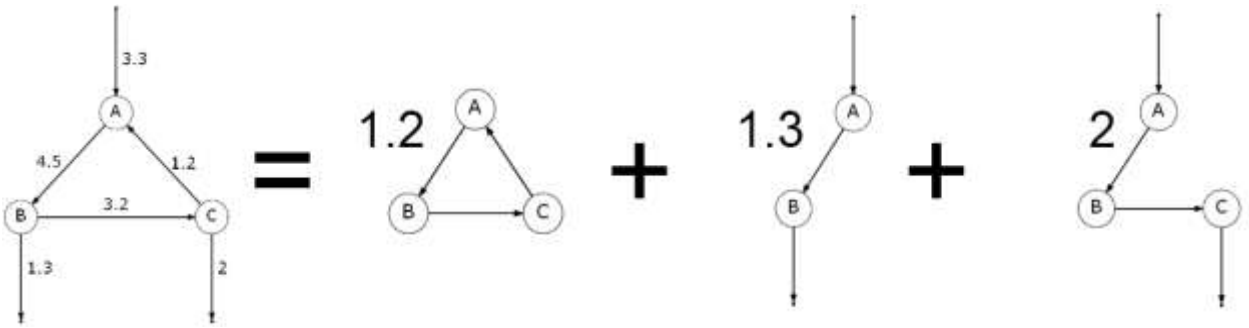


Figure 5: This figure illustrates a functional decomposition pictorially using fluxes. The flux coefficients can be distributed over the edges in their respective flux and then summed to recompose the original network.

As systems grow more complex, problems with this methodology arise. The methodology can still produce a valid functional decomposition, but the coefficient values for each of the fluxes become arbitrary. To illustrate this problem a new, slightly more complex system is introduced. This system is illustrated in Figure 6.

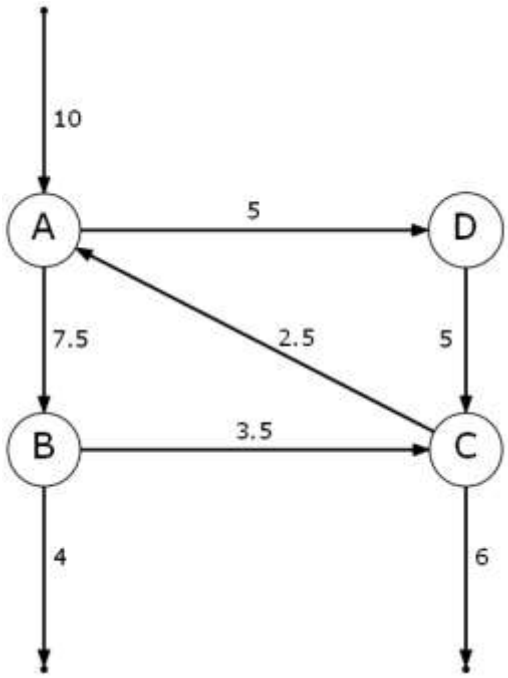


Figure 6: Second example system.

In this more complex example the edge with the least amount of flow is the edge between compartments A and C. This edge can be removed, but since it is included in two different cycles, a choice must be made about which cycle to remove first, as well as how much of that edge to remove with the cycle. Figure 7 shows that when removing the first cycle (the choice is arbitrary) any amount of flow greater than or equal to 0 and less than or equal to 2.5 can be removed with it. The second cycle would get a coefficient equal to the remaining flow left on that edge. The arbitrary nature of this process makes the coefficient vector produced less meaningful and undesirable for use in machine learning or data mining computation because no justification can be given to the order of cycle extraction or the amount of flow extracted.

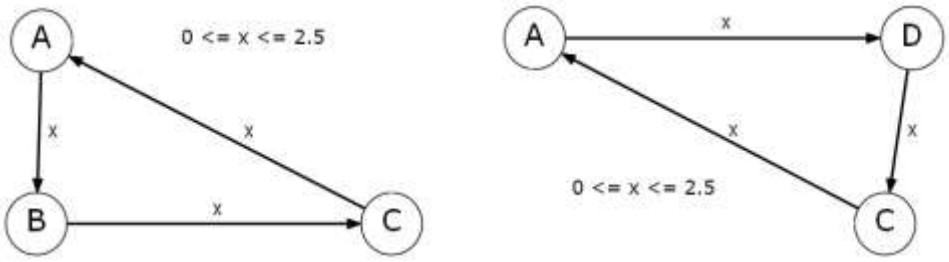


Figure 7: In the currently available methodology either of these two fluxes can be the first flux removed for functionally decomposing a system. The first flux removed would have a coefficient greater than or equal to 0 and less than or equal to 2.5.

The new methodology proposed by this work seeks to solve the arbitrary nature of the results produced by the process outlined above. The methodology proposed herein will simulate pathway data through a system and analyze that data to see how often fluxes are occurring in the simulated data. A pathway through a system is constructed by a certain combination of fluxes, and any pathway through the system can be parsed to reveal the fluxes which comprise it. There

is a probability for each flux to occur based on the inherent flow values in the system; for instance, if some agent is traversing a pathway through the system presented in Figure 5 and is at compartment B, it has a %53 chance of going to the environment (thus exiting the system, as the environment is a starting and terminal compartment for pathways) and a %47 chance of continuing on to compartment C. The choice at that step in the pathway determines which combinations of fluxes can be used to parse the pathway. The methodology proposed in this work analyzes large amounts of simulated pathways to leverage this probability throughout the system when calculating the coefficients produced in the functional decomposition. This strategy overcomes the arbitrary nature of calculating these coefficients and gives the functional decomposition meaning defined by the pathway traversal probability embedded within the nodes of the system. This probability mesh gives determines the values for the resulting functional decomposition vector, allowing it to be used for powerful data mining and knowledge extraction. The coefficient vector produced herein is a special point in the solution space of coefficient vectors and is not defined by the order of cycles removed or the quantity of flow arbitrarily chosen at a certain step. The methodology presented in this work lays out an algorithm that can compute this same meaningful point each time the algorithm is applied. Since the point has intrinsic meaning, it can be used powerfully in comparative analytics.

The ability to find the meaningful point outlined above, as opposed to an arbitrary point, is at the core of the originality in the work herein. With this newly created methodology, powerful machine learning can be accomplished for data mining and knowledge extraction towards complex systems.

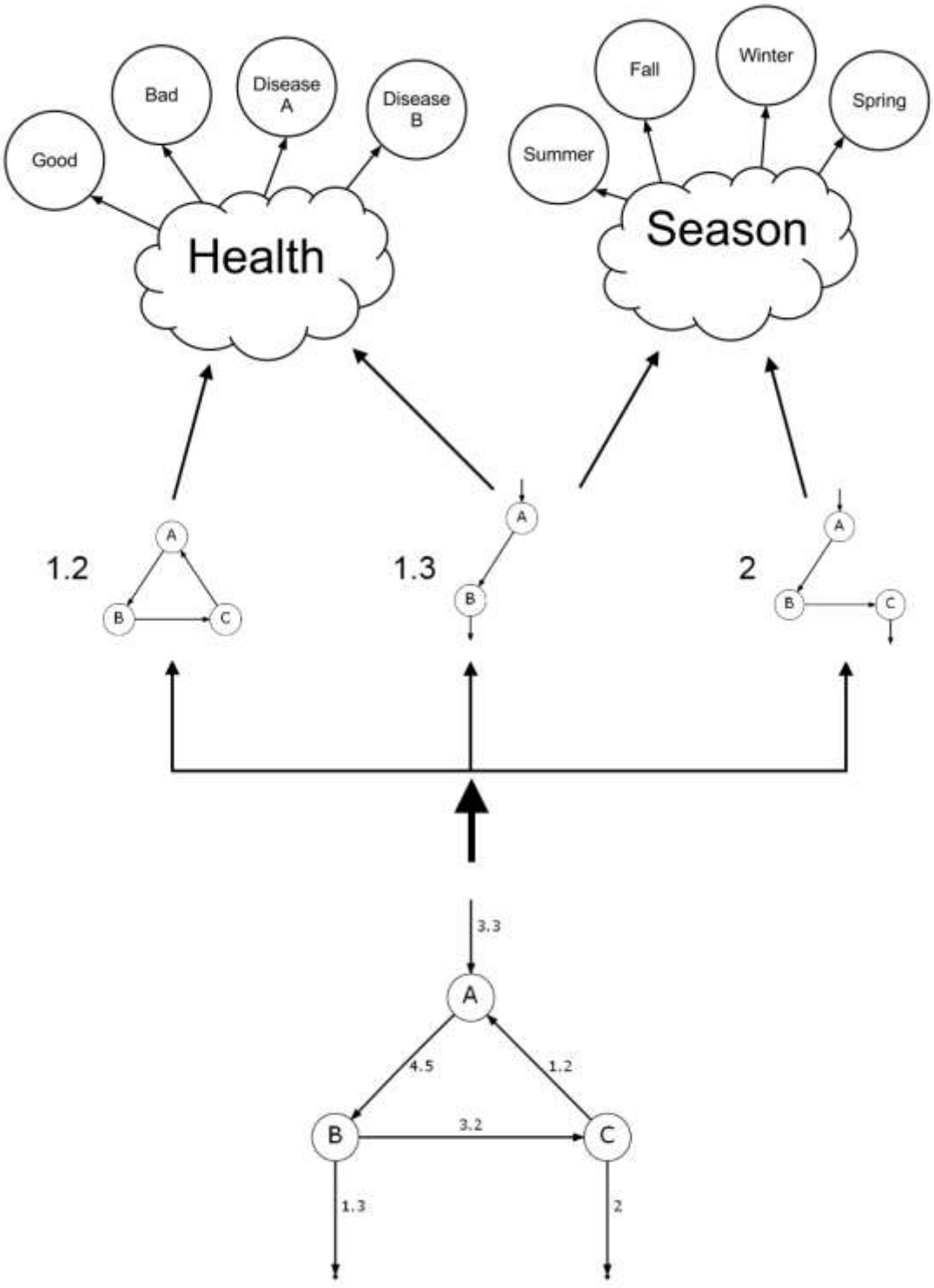


Figure 8: A high level diagram of how a system can be broken into compositional elements; subsets of those elements can then express certain property values at higher level system traits.

System DNA

The novelty of this work can be seen in the following analysis of the methodologies formulated herein. These methodologies present a framework that can be used to understand which low level compositional elements within a system encode information relating to higher level system traits. This kind of encoding framework is reminiscent of what genetics does for biological systems. In biological systems, nucleotides are basic components that get grouped together into codons, and these codons then appear together in groups to form genes. The values within these genes encode higher level traits like height, eye color, health conditions, *etc*. The analysis framework presented here also includes three levels of hierarchical groupings for compositional elements within complex systems. The base layer presented is compartments, similar to nucleotides from genetics. The secondary layer, similar to codons, consists of groupings of compartments called fluxes (simple cycles within a system graph). Finally, flux groupings within a system encode information relating to system traits like seasonality (fall, winter, spring and summer). These grouping of fluxes will be used as features in a machine learning methodology to see which groupings optimally classify system trait values for presented data sets. When seen in this light, groupings of fluxes become comparable to genes in biological systems. The ability to trace high level system traits back through these compositional layers to the building blocks of the system is powerful and only possible to date using the methodologies described herein.

CHAPTER 2

LITERATURE REVIEW

Due to the inherent nature of graphs to model problems, graph theory is a well-established field that spans many disciplines. Examples of the wide variety of problems incorporating graphs include facial recognition [5], Boolean function manipulation [4] and time series analysis [6]. Data mining is a younger field that has recently evolved into a well-established discipline [7, 8, 9, 10]. Together, these fields comprise a framework that can be used to model and analyze complex systems.

A system is a network of fully connected, interrelated components in which each component is related to every other component, either directly or indirectly [14]. In a complex system a network of components interacts nonlinearly, giving rise to emergent behavior [3]. Complex systems are effectively modeled as graphs. The analysis of complex systems can take on many forms finding a basis in fields such as fuzzy logic, graph theory, mathematics, nonlinear dynamics, *etc.* [1, 2]. Complex systems prove quite challenging to understand. Certain complex systems, *e.g.* food webs or power grids, are simply beyond current mathematical analysis, and to make progress with these systems one must suppress certain system properties in order to study other properties [2]. Machine learning, computational intelligence, genetic algorithms, *etc.* all present ways of dealing with complex, nonlinear problems having high dimensional feature spaces that lie outside the realm of computable mathematics [11, 12]. All these techniques are included within the more general field of data mining, a field that seems optimally poised to perform analytics on complex systems.

This section provides general insight into the current state of complex systems and data mining, then illustrates how complex system analytics can benefit from data mining. The section will be structured into two major parts relating to the aforementioned purposes.

An Overview of Complex Systems

The concept of systems has a rich history. The systems concept can be traced back at least to the 18th century [15], but sometime in the last century this concept became clearer and a generalized field of study emerged. Another author's paraphrased description [15] of the path by which the field came to be is that it was a clumsy, stumbling process that eventually came to fruition and now permeates the very nature of generalized science.

When discussing systems one encounters a multitude of other scientific disciplines such as physics, sociology, chemistry, biology, computer science, *etc.* This saturation across specialized disciplines is due to the power of systems theory to provide scientific interpretation and theory where none previously existed and with great generality [15]. This increased generality (higher level of abstraction) from specialized sciences defines the interdisciplinary nature of systems theory. Bertalanffy, in his General Systems Theory [15], proposed this statement as to the purpose of systems theory:

"Science has tried to explain observable phenomena by reducing them to an interplay of elementary units investigable independently of each other, conceptions appear in contemporary science that are concerned with what is somewhat vaguely termed wholeness, *i.e.* problems of organization, phenomena not resolvable into local events, dynamic interaction manifest in the difference of behavior of parts when isolated or in a higher configuration, *etc.*; in short systems of their respective parts in isolation."

In science, it is thus natural to isolate variables as far as possible, perform measurement and then interpret results. This process misses out on general properties of those variables that emerge only when all variables are studied as a whole, interconnected system. In order to fulfill its goals, systems research borrows from many different disciplines such as classical mathematics, computerization and simulation, compartment theory, set theory, graph theory, net theory, cybernetics, information theory, game theory, decision theory, and queuing theory [15, 16, 17].

The term "system" can take on different meaning, depending on the field of study under discussion. In linear systems theory, a system is something that takes input and produces outputs [18]. In biology a system is a group of components concerned with the same function (*i.e.* nervous system, digestive system) [19]. In ecology a system is a grouping of physical and biological components of an environment [34]. An overall description of a system is a grouping of interdependent, interconnected elements [19].

Bertalanffy [15] described two distinct types of systems, open systems and closed systems. Closed systems are isolated from their environment. Theoretically, they are allowed no interaction with their environment, but in actuality there are no completely closed systems, only ones that have varying degrees of interaction with their environment. Open systems, by contrast, interact freely with their environment. Bertalanffy [15] stated that certain systems by their very definition are open systems, for example living organisms. In order to survive, a living organism must interact with its environment for a number of reasons, one of which is to intake food. In closed systems physical chemistry can quantify reaction rates, dispersion, and the thermodynamic equilibrium eventually reached; in contrast, open, living systems never maintain equilibrium. Bertalanffy suggested these systems do reach a steady state, but this differs from

equilibrium. The term equilibrium is used in different fields, in statistical mechanics [20] an equilibrium state is defined as an average over all states in phase space consistent with the laws of conservation. Generally equilibrium defines a point at which competing influences have reached a balance point; as an example, if hot and cold water are mixed together in a glass, thermal equilibrium would be reached when all water in the glass reaches the same temperature (*i.e.* the hot and cold water no longer influence each other and reach the same temperature).

Entropy is a concept closely associated with equilibrium. Entropy is defined in statistical mechanics [20] as the gravitation of a distribution of atoms and molecules toward their most probable state. Entropy is sometimes viewed as a measure of disorder [17]. The result of entropy acting on a glass of water containing a divider keeping water dyed yellow on the right and water died blue on the left can be seen when the divider is removed. Once the divider is removed, there is no longer a barrier counteracting the external forces of energy acting on the glass, *i.e.* the earth's rotation, vibration, *etc.* Without this barrier the water molecules arrange themselves in the most probable configuration given the external forces acting on them, which is not to keep yellow on the right and blue left. The most probable configuration would be for the molecules to arrange themselves into a mixture of blue and yellow molecules that would optically represent a more greenish tone. This example illustrates a group of molecules containing more order (separated by color) but in an extremely improbable state arranging themselves with less order into a state of higher probability (an even distribution of yellow and blue molecules throughout the glass). The initial state of the molecules in this example displays less entropy and more order, and the equilibrium state displays more entropy and less order. This pull of entropy toward equilibrium gives entropy the label "the arrow of time".

Any system can be defined in terms of state variables, *i.e.* a set of quantifiable properties. Each different combination of state variables defines a system state. In a system at steady state these state variables can remain both outside of equilibrium and constant, despite the pull of entropy. For steady state to be maintained there must be some flow through a system. Bertalanffy [15] suggested the explanation for this apparent violation of entropy is the expanded entropy function of Prigogine. Bertalanffy stated that in a closed system entropy must increase according to the Clausius equation:

$$dS \geq 0$$

Formula 1

He stated that in an open system entropy can be offset by input into the system, and thus the formula for entropy in an open system is Prigogine's [21]:

$$dS = d_e S + d_i S$$

Formula 2

Formula 2 describes the entropy of an open system as the sum of both imported entropy ($d_e S$) and the production of entropy due to internal work by the system ($d_i S$). According to the second law of thermodynamics $d_i S$ must be positive. Open systems maintain their ability to resist equilibrium due to $d_e S$, which can be viewed as entropy transport. The term $d_e S$ can be positive or negative by importing matter with free energy (*i.e.* negative entropy).

If an open system reaches steady state it is independent of the initial state of the system or any perturbations along the path to steady state. In open systems steady state is reached and defined by the parameters of the system (*i.e.* rate of reaction and transport) [15]. This convergence principle is called equifinality. This differs greatly from convergence in a closed system where a final state and system inputs are concretely associated with one another.

Complexity within Systems

Because there is no agreed upon theory of complexity, distinguishing a system as complex is based more on intuition than definable characteristics [17]. In the absence of a generalized theory, one can only describe properties and traits that are generally present when a system is complex. A complex system is typically comprised of building blocks (*i.e.* atoms, molecules, cells, or agents), which can be complex systems themselves, giving rise to hierarchical levels of interaction. These complex systems interact with each other to exude actions, traits, behaviors, and organization not observable from studying the building blocks in isolation [19]. Complex systems generally contain many agents, and there are usually simple rules governing interaction amongst the agents. Typically, there is iteration over these rules leading to the distinguishable observations at a macroscopic level [17]. Complex systems behave non-linearly, and discovering the rules of interaction within any given system is difficult. An example of the complexity that can be derived from a simple set of governing rules can be easily seen in graphics produced from L-systems. An L-system is a type of formal grammar in the Chomsky hierarchy. Formal grammars are used to describe formal languages (sets of literal tokens and production rules governing their transformations). The literal tokens are agents in the system following interaction rules defined by the grammar, and in a graphic produced by an L-

system the grammar structure determines the composition of the image. An example of a simple grammar used to describe a language can be seen in Figure 9.

In Figure 9 a formal language is described that can generate the string 000#111. The capital letters A and B are variables in the grammar that can be replaced by elements on the right side of their respective production rules (*i.e.*, A can be replaced with 0A1 or with B, and B can

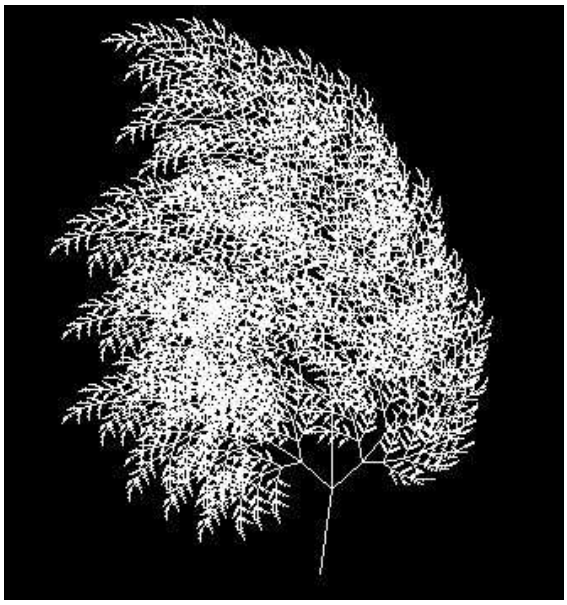
$$A \to 0A1$$
$$A \to B$$
$$B \to \#$$

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$

Figure 9: Example courtesy of Sipser [24].

be replaced with #). Figure 9 shows the derivation of the string 000#111 given the starting symbol of A. An L-system is formally described [23] as an ordered triplet $G = <V, x, P>$ where V is the alphabet of the system (containing variables and constants), x is the starting symbol and P is the set of production rules governing transition in the system. L-systems were applied to picture generation in 1984 when Aono and Kunii [25], and Smith [26] used them to create realistic pictures of plants. Pictures are created with L-system using turtle graphics. In turtle graphics the turtle is an agent on a plane that has a location and orientation. The turtle can be moved an indeterminate number of spaces along its orientation, and it can draw a line between its current location and the location it is moving towards (the turtle can also be moved without drawing a line). Given a specified starting state, an L-system can be used to encode a series of actions that can be iteratively applied to a two dimensional plane to produce an image. The system starts at some initial seed state and is iteratively expanded via the production rules from

the grammar a certain number of times. This final string is then used to command the turtle. An example of a basic implementation of L-systems graphics can be seen in Figure 10.



Alphabet

F	Move forward by X amount
+	Turn left by turning angle
-	Turn right by turning angle
[	Save state to stack
]	Pop state from the stack
|	Move forward by step amount

angle	8
X	100
step	0.5

Grammar

F -> |[5+F][7-F]-|[4+F][6-F]-|[3+F][5-F]-|F

* numbers in front of + and - signs indicate the number of times the sign is repeated

Start State

F

Figure 10: This illustration of a fern leaf can be made with an L-system using the parameters defined in the example. Image and example courtesy of James Mathews' software, L-Systems Explorer http://www.generation5.org/content/2002/lse.asp.

In Figure 10 one production rule and an alphabet of only six literals produce an image that is close enough to its real world counterpart to be recognizable. L-system based picture generation becomes even better at reconstructing real world objects as rendering techniques get more advanced. Figure 11 and Figure 12 are examples of L-system approaches that use more sophisticated rendering systems.

Figure 11: Image produced using an L-system based approach by Anastacio *et al.* [27].

Figure 12: Image produced using L-system techniques by McCormack [28].

These illustrations poignantly convey two important aspects of complex systems:

1. Complexity can be obtained through relatively simple rules of interaction governing a network of individual agents (grammar tokens in this case).
2. When studying a system at its most primitive level (in this case an alphabet, grammar and start state) it is difficult to see the system's general purpose.

Self-organization is a term central to the study of complex systems, and can be roughly (though not exhaustively) subdivided into four overlapping topics [17]:

- inherent computation
- non-linear and non-equilibrium processes
- emergence
- evolution

Inherent computation is the idea that interaction systems are governed by a finite set of rules, as seen previously in the L-system examples. Embedded in this concept is the notion that complex and ordered global patterns can develop as the result of local adherence to these fixed rules of interaction. This notion of interaction within a system is notable because it does away with a central processing unit. Instead of one entity being responsible for controlling the development and behavior of a system, every agent is responsible. Each agent locally obeys a finite rule set, thus creating mass parallelism in the computational work done in the system.

Non-linear and non-equilibrium processes are the general non-linear behaviors of a complex system which give rise to conditions within the system that reinforce irregularities into large scale patterns promoting decreases in entropy [17]. This topic is outlined by Ilya Prigogine [21, 29, 30], and deals with a system's ability to sustain non-equilibrium states (*i.e.* decreasing entropy) which put the system into unstable states where uniformity is dampened and new order is created.

Emergence is the idea that patterns (order) can develop within systems. Boolean networks are a good illustration of emergent behavior. A Boolean network is a group of interconnected Boolean variables, each of whose state is determined by function mapping from other variables in the network. The initial values for each of the nodes in the network can be randomly assigned, and then the iterations over the network are performed such that for each Boolean node x in the network there is a fixed set of nodes $V = (y_1, y_2, \cdots, y_n)$ whose combined state is mapped through a function *F(V)* to determine the value of *x* at time interval *t + 1*. The value at the current time interval is used for each node in the network, x_t, to determine the values for the next interval such that $x_{t+1} = F(Vt)$. Bossomaier [17] explained that, since there are a finite number of possible states, any Boolean network must fall into a fixed cycle of states called a limit cycle. Since the network is deterministic, this limit cycle will always be reached, and it will repeat itself. This pattern of network states is an example of emergent order from a system.

Evolution within complex systems is the idea that systems can change over time to adapt to environmental influences. In biology this term is used to account for the development of varying species, but that usage is only one instance of the more general concept that systems can change. Connectivity illustrated in a directed graph is a representation of interaction between

elements within a system and illustrates the concept of evolution within systems. Bossomaier [17] explained two key concepts in this regard: First, he posited a theorem that matrix models (*i.e.* linear systems and Markov processes), dynamical systems (*i.e.* differential equations), cellular automata, semi groups, and ordered sets are all isomorphic to directed graphs. Second, he stated another theorem that in an array of automata, the state space forms a directed graph. According to Bossomaier these two theorems show that the properties of directed graphs are inherent in both the structure and behavior of all complex systems.

If directed graphs are inherently at the heart of complex systems certain properties of directed graphs become intriguing, one of which is the changes in connectivity of randomly created directed graphs. Green and Bossomaier [17, 31, 32] mentioned that when directed graphs are randomly generated there comes a point at which adding more edges drastically increases the connectivity within the graph. This point is sometimes called the double jump point. Such phase change is sometimes called the edge of chaos, as systems near this point exhibit increasing signs of chaotic behavior. It is thought that from an evolutionary perspective this increase in the chaotic nature of a system strengthens the system's ability to adapt, thus giving it a selective advantage [17].

An Overview of Data Mining

Humans have analyzed data for a long time. Bayes' theorem traces back to the 1700s, and is currently a fundamental concept behind certain forms of data mining [38, 39, 40]. With the advent of digital computers, the amount data, and speed with which it can be analyzed, drastically increased. This created a need for new forms of data mining. Data mining, defined herein as generally as possible, is nontrivial extraction of implicit, previously unknown and potentially useful information from data [35, 41, 42]. The work herein concerns knowledge

discovery for propagation of useful information to form a more complete understanding about the behavior of an entity.

Given the large amount of digital data present in the world today, data mining can produce a flood of information with varying degrees of usefulness. Knowledge discovery is concerned with finding interesting, useful patterns in data [7, 35] and making them understandable to humans. Mitra [41] stated that knowledge discovery is the process of turning low-level data into high-level knowledge. He stated that d*ata* are a set of facts *F*, and a pattern is an expression *E* in a language *L* describing the facts in a subset *FE* of *F*. He called *E* a pattern if it is simpler than the enumeration of all facts in *FE*. He stated that pattern validity is measured by a function *C* mapping expressions from *L* into a partially ordered measure space *MC*. He also introduced functions for mapping the usefulness (*U*), novelty (*N*), and understandability (*S*) of an expression into numeric feature spaces. He then introduced one more measure called Interestingness, *I = (E, F, C, N, U, S).* *I* is subject to some user defined threshold to proliferate only interesting patterns to the user. From here Mitra made the intriguing point that a measure of interestingness can be designed using two different approaches, objective and subjective. The objective approach has the benefit of generality and applicability across any pattern in any knowledge discovery process, but this approach may fail to capture certain patterns that can only be found using contextual clues from the particular domain being explored. Subjective approaches employ user-defined context to find patterns that objective criteria may miss.

Behavior mining is a new concept introduced by Chen [36]. Chen posited the idea that data is a simplified way of abstracting the behavior of some entity, and that data sets don't simply contain numbers and tokens, but an underlying behavior. Data stored in relationally formatted tables can help in understanding the underlying behavior of an entity, but this simplistic form of

data can be limited. As an example take a data record consisting of the metadata from a file on a computer. Data such as the author, file size, data modified, or title, ignores the meaning embedded in the actual text of the file. The text of the file contains more knowledge and is richer in information; however it is also more complex and thus harder to mine with computational methodologies. Much progress has been made towards the processing of complex data forms such as sounds files, images, and text [43, 44, 45] which allows data mining to explore rich fields of information, paying closer attention to the semantics buried within these complex data forms. Without respecting the semantics of the information being processed, the scope of information that can be mined becomes limited. It is in the semantics of these complex data forms that the true behavior of the underlying entity is described [36].

The discovery of knowledge alluding to the underlying behavior represented by data is a goal that machine learning, evolutionary computation, and computational intelligence have progressed to fulfill. While not exhaustive the following sections will cover some of the existing algorithms in these fields. There is overlap between these fields so algorithms that could be placed in multiple fields will be placed in the field with which they most correspond.

Machine Learning

According to Mitchell [11] machine learning is a broad field in computer science that attempts to create algorithms that learn by experience. Inductive learning does not fit into some people's definition of acceptable computer science, but its effectiveness at solving a wide variety of problems that standard numerical procedures cannot solve is indisputable. Machine learning algorithms attempt to learn a concept from a distribution of training data. There are two general types of machine learning, supervised and unsupervised. In supervised learning the learner is given examples and told what class of data the example represents. In unsupervised learning the

data instances are presented to the learner without this classification knowledge. Machine learning techniques are applied to data mining in a variety of ways, including automated classification systems, pattern recognition, and rule discovery.

Support Vector Machines (SVM) are a type of supervised learner, and they have recently garnered much attention in the machine learning community for their ability to work with high dimensional, nonlinear data. The strength of the SVM's is in their ability to divide a feature space into classification regions using a maximum marginal hyperplane (MMHP). A MMHP is a plane that divides a feature space so that the plane is equally distant from both classification boundaries. Such a division gives the hyperplane the maximum distance from both classification instances, thus allowing the SVM to generalize optimally given the supplied training data distribution. Research done by Muller *et al.* [46] applied SVM's to forecasting time series data, and the SVM's outperformed existing methods significantly (in some instances improving on the best known methods by a measure of 29%). Research by Burges [47] applied SVM's to generalized pattern recognition. The SVM learners are useful because their classification and function approximation capabilities make them versatile. Additionally, they have been known to outperform other learners on a variety of problems due to their superior ability to generalize and deal with higher dimensionality [51, 52, 53].

Neural networks are a type of machine learning algorithm inspired in part by the neuronal network of the brain. They are adept at learning real valued, discrete valued and vector valued functions from examples [11]. These networks usually contain a grouping of interconnected nodes that take one or more inputs, aggregate the input values and then pass an output response to another node in the network. They are trained as supervised learners using some kind of training algorithm, *e.g.*, back propagation using gradient descent, to fit a distribution of training

data. The architecture in a neural network is dependent on both the semantics and classification of the problem it is being applied to; for example, recurrent neural networks are commonly used in time series learning architectures. These types of networks are typically back propagation networks that use additional first layer input nodes in their architecture. These additional inputs are values that have been saved from the previous run of the network and act as context for the current set of values. Typically, the values from the hidden nodes or the output nodes in a network are the values that are saved and sent through these additional input nodes.

Typical recurrent network structures are the Jordan and Elman architectures. In the Elman architecture the values from the last iteration of the hidden nodes are sent through the network as inputs. Research by Cai *et al.* [50] used Elman networks for time series prediction, and the researchers obtained a high degree of accuracy. In a Jordan network, the values from the output layer are fed as inputs into the network. Jordan and Elman networks work well for forecasting time series data because they exploit the inherent relationship between sequential time series data instances in order to provide a context to the activity of the time series. This context gives extra insight, thus enabling a more accurate forecast.

Self-Organizing Maps (SOM) are a type of neural network that perform unsupervised clustering of data distributions [48]. Tak-chung Fu *et al.* [49] used the SOM architecture to find recurring patterns within stock market time series data. The typical network architecture for an SOM is either a one or two dimensional array of output nodes which are used as centroids for clustering. Training an SOM involves initializing each of the nodes in the output node structure with random weights. These nodes, which could more accurately be described as centroids, are thus placed randomly throughout the feature space. As input vectors are sent into the SOM for training, the vector belongs to the centroid closest to the vector. This centroid is relocated

slightly closer to the vector in the feature space using a learning parameter (alpha). In addition to the winning centroid, a neighborhood of centroids around the winning centroid in the two-dimensional output node structure are relocated, to a lesser degree, towards the training instance vector. Multiple incrementations of this process are run until a stopping criterion is met.

A useful feature of SOM's is that they are able to reduce the dimensionality of classification problems while maintaining the topological structure of the feature space. A typical example used to describe the topological preservation tendencies of an SOM is to apply it to the relatively easy problem of classifying colors in an RGB color space. The feature space for this problem has three dimensions, but a typical SOM for this problem would have a two dimensional output node structure. After training the SOM, the centroids in the output array would assume color values that are similar to their neighbors (*i.e.* shades of red would be located near each other and they would also be located near shades of orange).

SOM's are useful in data mining because they allow users to explore and analyze data when specific classifications and structure are not known. Research mentioned above, by Tak-chung Fu *et al.* [49], applied SOM's to time series analysis by using them to search for recurring patterns in time series without prior knowledge of the pattern being sought.

Evolutionary Computation

According to Eberhart and Shi [12] the two main areas of application for evolutionary computation are optimization and classification. These strategies are highly effective for solving combinatorial optimization problems. Typically they involve a population of individuals that are moved throughout a search space in a random but controlled manner. The population of individuals represents potential answers to a problem, and they are assessed some fitness value that corresponds to the strength of their solution. The schemes for moving populations around

the search space vary, and this gives each strategy certain strengths and weaknesses for solving different problems. There is not one best evolutionary algorithm which is used for all problems because the search spaces for individual problems can present qualities that are better suited for certain strategies. Interestingly, this family of algorithms is heavily inspired by nature, and evolution and swarm intelligence provide two examples. A survey paper compiled by Freitas [54] outlined how this family of algorithms is commonly used in data mining for problems such as classification, feature selection, feature generation, and clustering.

The ability to evolve is a biological tool that promotes survival and ways for organisms to adapt to an ever-changing environment. Holland [55] described evolution as a powerful problem solving mechanism that should be harnessed for use rather than just envied. The Genetic Algorithm (GA) is modeled after this evolutionary process, encompassing survival of the fittest and natural selection through the use of selection, crossover, and mutation operators. Holland [55] presented an explanation of how the GA samples a search space, building schemas (combinations of alleles) that increasingly get closer to the global maximum as iterations of the search process are run through the schema theorem. He mentioned in this theorem that schemas with bad fitness should decrease but schemas with good fitness should increase exponentially. The Schema Theorem is formally defined as follows:

$$m(H,t+1) \geq \frac{m(H,t)\, f(H))}{a_t}[1-p]$$

Formula 3: Schema Theorem

GA's can be used for data mining in different ways. One area in which they are used substantially is classification. Classifier systems evolve sets of rules that classify data instances.

The ultimate goal of these systems is to discover the best set of rules to classify data. GA's are also frequently used for feature selection which can be an important part of the data mining process. Typically, data mining algorithms compute features from data that allow the algorithms to make inferences about data instances. The common problem with this methodology is knowing which combination of features best represents the ultimate purpose of the algorithm. Due to their ability to search for globally optimal solutions, GA's present an effective way to optimize the set of features used in the data mining process. Finally, an interesting use of GA's for data mining on time series data is presented in research by Povinelli [56]. In his work he represented a feature space as an augmented phase space and then used a GA to optimally cluster the set of time series points in the augmented phase space. This algorithm clustered the time series points into groups that represented general events in the time series data.

Genetic Programming (GP) is an interesting technique that resembles GA's, but the individuals being evolved to optimize a problem are not answers. In GP the individuals are algorithms represented as parse trees. Eads *et al.* [57] used GP to classify lightning in the atmosphere into different categories. The researchers used GP to generate features on the time series data collected from weather satellites. The GP algorithm had specific operations that it could combine into algorithms that computed features on the data. Once the feature generation algorithms were created, they were used in combination with SVM's to classify the data input.

Particle Swarm Optimization (PSO) was initially presented by Eberhart and Kennedy in 1995 [58]. It is a global optimization strategy inspired by the social aspects of bird flocking and fish schooling. In PSO a population of individuals is moved through a search space using inertia and velocity. An individual has a bias to gravitate towards its best position in the search space as well as the position of the best solution found by any individual in the swarm (the global best

solution). These bias factors are controlled by local and global constants *c1* and *c2*. The population is moved through the search space using iterations of the algorithm until the population converges on a best solution. PSO algorithms have been effectively applied to clustering data in research by DW van der Merwe *et al.* [59]. The researchers used PSO clustering techniques on six different data sets, and the PSO showed better results than standard K Means clustering in this research. PSO is a recent trend that shows promising results for certain optimization problems. It can effectively be used to enhance data mining in combination with more standard machine learning approaches. An example of this enhancement is a trend to optimize the weight variables in a neural network using PSO algorithms.

Ant colony optimization is a population-based search strategy that models the ability of an ant colony to find the shortest path to food sources from their nest. The ants drop pheromones as they walk, and as more ants follow a path, an increasing amount of pheromones attracts more ants. Initially, if an ant comes to an obstacle it will have a 50-50 probability of choosing either way around it; obviously, the shortest way around the obstacle will provide a faster way for ants to reach the end goal. In a given time window more ants will be able to travel the shortest path, which will ultimately build stronger pheromone levels depicting the optimal way to the food source. Research by Parpinelli [60] showed the usefulness of ant colony optimization towards the data mining goal of classification.

Computational Intelligence

Computation Intelligence is a sub-field of artificial intelligence. It combines methodologies such as Fuzzy Systems, Neural Networks, Swarm Intelligence, and Evolutionary Computation. Computational intelligence typically uses hybrid approaches to solving complex problems and combines these different methodologies into one system.

Affinity propagation is a clustering technique recently proposed by Frey *et al.* [61]. The idea behind this algorithm is that data points are placed into a search space and messages are passed iteratively back and forth among them with the ultimate goal of finding optimal exemplars (representative points for a group of points) and points belonging to those exemplars. Messages come in two different forms. The first type of message is passed from a data point *i* to an exemplar candidate *k* and represents how well suited *k* is to serving as the exemplar for *i*. The second type of message is sent from exemplar *k* to data point *i* and indicates how appropriate it would be for *i* to choose *k* as its exemplar. This algorithm iteratively passes messages back and forth between data points and eventually well-defined exemplars are produced that represent other data points. Although this algorithm is relatively new, it has been applied to a variety of problems such as clustering face images, detecting gene expressions, identifying key sentences between drafts of literary work, air travel routing, *etc.* The speed with which the algorithm works makes it intriguing and the results produced are on par or better than competing methods in the experiments provided by Frey *et al.* [61]. A major benefit of this algorithm is that the number of exemplars (clusters) does not need to be known beforehand; the algorithm can figure out through message passing how many exemplars are needed and how to partition the data instances.

Artificial Immune Systems (AIS) were developed through research by Packard and Perelson [135] in 1986. Different representations of AIS's exist, and they all attempt to model the biological immune system. Examples of AIS systems applied to pattern recognition can be seen in research by Wilson *et al.* [62] and Hunt *et al.* [63]. Wilson *et al.* [62] presented a particularly eloquent methodology in which no prior knowledge of the pattern being sought needs to be known. The general flow of an AIS is to maintain a population of B cell objects

responsible for recognizing certain antigens. Certain B cells respond to particular antigens, and the strength of the association between the B cell and the antigens determines the level of stimulation the B cell receives. The more stimulation the B cell receives the more likely it is to proliferate itself in the population of B cells.

The general flow of the algorithm as applied in research by Hunt *et al.* [63] is as follows:

```
Randomly initialize B cell population
Load antigen population
Until termination condition is met do
   Randomly select an antigen from the antigen population
   Randomly select a point in the B cell network to insert the antigen
   Select a percentage of the B cells local to the insertion point
   For each B cell selected
      present the antigen to each B cell and request immune response
   Order these B cells by stimulation level
   Remove worst 5% of the B cell population
   Generate n new B cells (n equals approximately 25% of the population)
   Select m B cells to join the immune network (m is approximately 5% of population)
```

Fuzzy C Means Clustering is a hybrid clustering strategy that incorporates fuzzy logic into standard clustering methodology. In Fuzzy C Means Clustering, data instances have a degree of membership to their corresponding clusters and they can belong to multiple clusters. The summarized steps for the Fuzzy C Means Clustering algorithm as outlined by Bezdek *et al.* [64] are:

```
Choose initial parameters and initialize membership matrix
LOOP
        Compute means (cluster centers)
        Update membership matrix for each data instance
        Check stopping criteria
```

Conceptually, this procedure fuzzifies the K Means algorithm so that a more complete understanding can be built from complex multivariate data sets. Fuzzy C Means Clustering can be a powerful data mining tool because it allows for a fuzzy understanding of data.

A common trend in computational intelligence is to use a global optimization strategy to train neural networks. One way to train standard multi-layer perceptron networks is to use back propagation, a gradient-descent strategy that can easily get stuck in local optima. Global optimization strategies can be applied to the training process to reduce the likelihood of becoming trapped in local optima. Research done by Cai *et al.* [65] used a hybrid PSO/Evolutionary Algorithm to optimize the weight parameters in Recurrent Neural Networks. The neural networks trained with the PSO/EA optimizer perform well.

Another example of this strategy is in research done by Rivas *et al.* [66]. The researchers trained RBF networks using a method resembling a GA. One reason this optimization strategy for training neural networks is beneficial is that creating a neural network and training it combines art and science. When building a neural network, parameters must be chosen without scientific justification for the selection. These parameters include the number of hidden nodes, number of input node (in some cases), activation functions (there can be valid scientific reason for specific activation functions, but not always), and network topology. Since the vast number of combinations for these parameters cannot all be evaluated, the optimizer offers help. As

mentioned earlier, optimization algorithms are beneficial also because training neural networks with methods like gradient descent can lead to a local optima rather than a more global one.

Applications in Data Mining

As data mining develops, researchers are using it to mine more complex data for more complex knowledge. This section will review some of the more recent progress in data mining as well as some less recent developments.

Tak-chung Fu *et al.* [49] used Self Organizing Feature Maps to discover patterns in stock exchange data. Because of the unsupervised nature of their learning style, SOM's can be used effectively for pattern discovery, since the learner has the flexibility of not knowing the classifications for the training data the SOM needs to learn. Additionally, the exact number of clustering centroids needed for classification does not have to be known a priori. Not having to know the number of different patterns being sought increases flexibility in the discovery process. The approach taken by Tak-chung Fu *et al.* [49] featured a two-dimensional output layer to classify instances of data. To obtain instances for classification, a sliding window of length *w* was used to break the stock exchange data into overlapping segments such that $W(s) = \{ si = (si,....,si + w - 1) \mid i = 1, ... n – w + 1 \}$. These instances were fed into the SOM and grouped to centroids which represented similar patterns of data. To make the SOM more effective, a post-processing methodology was used which filters unused output nodes and groups output nodes within a certain distance together in order to reduce redundancy. The ultimate result of training the SOM was that output nodes represented a group of similar input patterns; together, they represented a recurring pattern. Tak-chung Fu *et al.* [49] went on to introduce a new pattern-matching scheme to enhance the clustering procedure. The general idea of the algorithm, referred to as PIP in the paper, was to describe the segments of data by important points that

reflected the general tendencies of the data. The proposed algorithm initialized the set of important points to the first and last points in the data segment, then incrementally added the next point that was the farthest distance away from the connecting line between any two important points. This process was run until the desired number of important points was located. This process generalized the data, effectively re-sampling it at highly descriptive points; the more points used, the higher was the resolution of the re-sampled data. Once the important points were captured, a similarity measure of direct point-to-point comparison was used to measure the distance between the data. The formula used to compute similarity was as follows:

$$Dist(I, SOI) = 1/n$$

Formula 4: Similarity computation formula. *I* is an instance set of important points and *SOI* is the group of important point sets.

Extensions of this formula can be implemented to take scale (time dimension) into account, and this extension was outlined in the work presented by Tak-chung Fu *et al.* [49]. The proposed algorithm has some interesting benefits that were mentioned in the paper. The re-sampling performed to find important points can substantially reduce the amount of time needed to train the SOM, if the SOM is trained on the sampled version of the times series rather than the raw data. Another benefit of re-sampling is that it provides an easy means of performing multi-resolution comparison of data segments. Different lengths of raw data segments can be sampled down to the same resolution and compared.

Povineilli *et al.* [56] presented a generalized methodology for detecting events. An event in time series data is application specific, but an example would be an earthquake in time series data of seismic activity. To detect events, the framework presented first called for the definition

of a phase space for the problem. Phase space, dating back to work by Willard Gibbs in 1901, is a way to represent a problem so that any degree of freedom or variable in a problem is seen as a dimension in a feature space. The feature space winds up being *n*-dimensional and is referred to as a phase space. For time series event detection, Povinelli [56] suggested what he referred to as an augmented phase space model. Any phase space can be made into an augmented phase space by simply adding another dimension to the phase space, representing what the Povinelli referred to as the degree of "eventness". Eventness was described as how indicative a time series segment is of an event happening and is represented by a function *g(t)*. Temporal pattern clusters were made within the phase space (not the augmented phase space) in which each cluster was a hypersphere centered on a given temporal pattern. The dimensions were equal to the phase space, and the radius of the hypersphere was an arbitrary parameter representing distance in the real metric space of the hypersphere. Each temporal cluster had an average *g(t)* value and the complement of every temporal cluster (every pattern not in the cluster) had an average *g(t)* value. The optimization performed by the GA was to find the optimal temporal clustering to predict events. The fitness function used was provided in Formula 5.

The GA maximized the value of Formula 5 to find the optimal temporal clustering for determining events in the time series data. As the average value of the set *M* grew and the

$$f(P) = \frac{\mu_M - \mu_{\overline{M}}}{\sqrt{\frac{\sigma_M^2}{c(M)} + \frac{\sigma_{\overline{M}}^2}{\sigma(\overline{M})}}}$$

Formula 5: Fitness function for algorithm presented by Povineilli *et al.* [56]. *M* is the set of time series segments in the temporal cluster, *c(M)* is the cardinality of *M*, μ is the average *g(t)* value and σ is the variance of *M* from μ.

average value of its complement shrunk, the top of the fraction grew larger. The bottom of the fraction shrunk when the variance of the set *M* and its complement was lower and the cardinality was higher. The desired effect was to make the top of the fraction grow and the bottom shrink resulting in the maximum value for the function being optimized. This effect was achieved when the average *g(t)* value of *M* was at its highest, the average *g(t)* value of everything not in *M* was at its lowest, the variance of both *M* and its complement were at their lowest, and the cardinality (number of elements in the set) of *M* was large. The resulting temporal clusters were determined to be events, and the GA searched for them. The chromosome representation for each individual in the GA was a standard bit string representation where groupings of bits represented possible values for each dimension in the phase space.

Wavelet Theory is an emerging methodology in data analysis. Wavelet Theory utilizes small wavelets to describe a larger function. Wavelets themselves are functions that have a zero mean and are localized in both time and frequency [6]. The Wavelet Transformation can be compared to Fourier Transformation; however, there are some distinct differences. One difference between the two is that the Fourier Transform treats a signal as a composition of sine and cosine waves, but the Wavelet Transform treats a signal as a composition of a more general classification of functions called wavelets. Another difference between the two is the resulting data produced from their respective transformations: The Wavelet Transformation contains a localized sense of time which can be of great use in data mining. Research presented by A. Grinsted *et al.* [67] used a Continuous Wavelet Transform representation of geophysical time series data, in combination with Cross Wavelet and Wavelet Coherence operators, to find correlation between different but related time series. The idea was to examine potentially linked time series data to infer causality between the two series. The data used in the experiments was

geophysical data; specifically, the data was the Arctic Oscillation measurement (AO) and the Baltic Sea ice extent (BMI). The AO represented the mean winter condition in the arctic atmosphere and the BMI represented winter severity represented by ice conditions [67]. The researchers used a Cross Wavelet operator in the hopes of discovering correlated phase lags due to causality between the two time series in the experiments. The Wavelet Coherence was then performed and the statistical significance of the coherence was estimated using Monte Carlo methods.

Feature selection can be a difficult task in data mining. Eads *et al.* [4] proposed a method of intelligent feature generation using Genetic Programming (GP). GP is an evolutionary computational method. It is related in form to the GA, but the individuals in the population being evolved are not possible answers to a problem; they are algorithms themselves. The individual in a GP is a representation of a parse tree for a computer algorithm. The population of individuals gets evolved until the best parse tree can be found. The data presented to the learner by Eads *et al.* [4] were spectrograms created from the Fourier Transform on radio waves traversing the ionosphere. When lightning in the ionosphere occurs the sensor equipment on a weather satellite sends the radio wave data to ground stations for recording. These transmissions were transformed using the Fourier Transform into spectrograms which were further processed into a power density series. There were 3181 samples in each power density series. The job of the learner was to categorize the spectrograms as one of seven types of lightning. There were two approaches to this in the paper. One approach was simply to throw the raw data into an SVM for classification, and the other approach presented was to use GP to evolve intelligent features from training data and then send those features into an SVM for classification. The GP approach used an internal SVM in the fitness function, and the fitness score for each individual

was computed by sending every spectrogram in the training data through the parse tree of an individual to produce a feature vector of *n* elements. The feature vectors were then sent through the SVM, and the fitness of an individual corresponded to the performance of the SVM on the training data using the supplied feature vectors. The first model for this operation used one SVM for the fitness calculation and was susceptible to over-fitting. The other model used a cross fold validation scheme that was more robust, but also took substantially more time to run. Once the GP evolved the feature-calculating algorithm, another SVM was used to evaluate the effectiveness of the model. During this test tenfold cross-validation was used to train the SVM. Once the SVM was trained, ten percent of the data that was withheld was used to evaluate the overall effectiveness of the model. The results showed that the intelligent feature creation scheme using GP did not perform as well as the raw data into the SVM. The percentage of data classified correctly with the intelligent features was 61.54 % while the raw data was able to achieve 70.38 %. While the results were slightly less using intelligent feature generation, the methodology presented was still novel and worth further evaluation.

Artificial Immune Systems (AIS) were developed through research by Packard and Perelson [135] in 1986. Different representations of AIS's exist all attempting to model biological immune systems, typically by modeling the memory scheme immune systems use to recognize pathogens.

An understanding of AIS applied to pattern detection can best be ascertained through work proposed by Wilson *et al.* [69]. These researchers utilized an AIS system to find motifs (patterns) in time series data. This methodology was unique in its ability to have no prior knowledge, either of patterns being sought or of their lengths. The implementation made use of what were deemed trackers, which contained a variable length of symbols used to represent a

dimensionally reduced equivalent representation of a time series subsequence [69]. The trackers were ultimately used to find string representations of recurring motifs in a series. The basic idea behind this algorithm was the transformation of a time series into a symbolic representation using Piecewise Aggregate Approximation. Once the series has been transformed, the subsequences were stored in a matrix; the AIS methodology then mined this symbolic representation for time series motifs. The trackers were initialized and matched to this symbolic matrix of time series data. Initially, the trackers were one symbol in length, but throughout the iterations of the algorithm, if a tracker matched some series in the matrix of series, it was given the potential to grow. This process gave the AIS the ability to find unknown patterns of variable lengths. A match was determined between a tracker and the time series symbol matrix using string operators in combination with a matching distance threshold. Every match a tracker made to the symbol matrix increased its count for the specific iteration of the algorithm by one. Trackers with a count greater than one were considered reoccurring matches.

A final note about the implementation of this algorithm is that trackers were grown and mutated in a controlled fashion after each iteration of the algorithm. This process allowed the algorithm to explore more of the problem's search space and ultimately find more complex patterns than it would have been able to if mutation and growth were not performed.

The AIS algorithm presented by Wilson *et al.* [69] was tested on two different data sets. The first data set was a benchmark data set dealing with steam generation. This data set has been used in other experiments by Keogh *et al.* [70], who used a probabilistic algorithm approach to pattern discovery. The results from the AIS algorithm were compared to the other published results. The two methods find the same motifs in the data; however, Keogh *et al.'s* method returns a slightly longer version of the dominant pattern. Wilson *et al.'s* method can be

considered to outperform Keogh *et al.'s* method since the former found this pattern without prior knowledge of the length. Keogh's probabilistic method requires the length of the pattern being sought which is not always known. The AIS algorithm used by Wilson *et al.* also generalizes much better to motifs of different length than does Keogh's method. The other data set used in the experiments by Wilson *et al.* was a power consumption data set. The AIS system produced significant results on this data set; proving itself sensitive enough to distinguish patterns of power consumption on a bank holiday weekend from the rest of the weekends in the data.

CHAPTER 3

METHODOLOGY

This research uses computational algorithms to decompose a network into smaller sub-networks, and then it computes a fraction of overall network flow that can be attributed to those sub-networks. This decomposition yields a more meaningful level of abstraction in network analysis than a simplistic focus on individual nodes and flows by providing contextual information about the importance of nodes in functionally relevant sub-networks. Using graph theory descriptors and constructs such as adjacency matrices, simple paths, and cycles, this research treats any network as a generalized, directed, weighted graph. Once a network is decomposed and all sub-networks enumerated, the second part of this analysis computes the fractions of total flow to attribute to each sub-network. These flow coefficients are grouped into a coefficient vector. While the set of sub-networks for a decomposed network is unique, the coefficient vector detailing their flow contribution within the total system is not. An averaging technique for dealing with this non-uniqueness will be introduced in order to compute a meaningful point in the output vector's solutions space.

Ultimately, this research transforms an adjacency matrix representation of a network and a corresponding distribution of flow data over that network into a set of sub-networks and a coefficient vector. This allows for novel data mining and optimization in the field of network analysis and has potential to be a unifying solution to a large problem set.

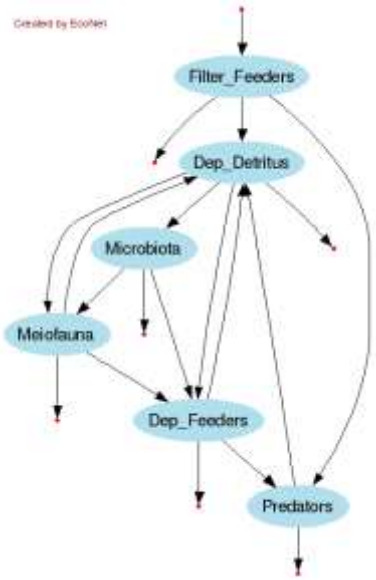
Computational Analysis of Network Subunits

When analyzing a network for regions of importance, metrics such as connectedness and throughflow are typically used. These measurements, while providing useful information, do not provide a high level summary of the general workings of a network. They comprise a low level depiction of the network that can lack contextual meaning and tend to be confusing when applied to the network as a whole. In this research a computational methodology is presented that allows a higher level understanding towards the workings of a network. It provides a general summary of activity over well-defined neighborhoods (referred to as sub-networks) within a network. First, a network is decomposed into the smallest possible sub-networks, and then coefficients are found that attribute certain portions of throughflow in the network to individual sub-networks. The coefficients for each sub-network represent the amount of flow over the entire network that each sub-network is responsible for. Given this higher level view it becomes possible to analyze nodes and regions in a network with more contextual information. This method provides useful information that enables powerful data mining on a wide range of symbolic data.

Introduction

A network is a set of interconnected compartments with directed edges (currency flows, *e.g.*, mass, energy, dollars, information). This type of network construct appears across a wide variety scientific disciplines [73, 76, 77, 78, 79] including economics, computer science, ecology, biology, and sociology. Network analysis historically involves the evaluation of network structure and function through the calculation of such metrics as nodal connectedness or compartment and total system throughflows. This approach can be helpful, but it lacks the ability to analyze groupings of connected nodes interacting with each other. Perhaps a more complete method for analyzing a network includes accounting for a node's sphere of influence

within well-defined sub-networks. This approach can serve to contextualize the behavior of a specific node and provide a more complete understanding of its role to the network as a whole. The goal of this research is to quantify a measure of flow (signifying importance) over groupings



	Environmental input	Environmental output
Filter Feeders	41.4697	25.1309
Dep Detritus	0	6.18326
Microbiota	0	5.76412
Meiofauna	0	3.58234
Dep Feeders	0	0.422359
Predators	0	0.387494

	Filter_ Feeders	Dep_ Detritus	Microbiota	Meiofauna	Dep_ Feeders	Predators
Filter Feeders	0	0	0	0	0	0
Dep Detritus	15.7567	0	0	4.24377	1.87502	0.350235
Microbiota	0	8.1778	0	0	0	0
Meiofauna	0	7.2803	1.20689	0	0	0
Dep Feeders	0	0.5983	1.20689	0.66143	0	0
Predators	0.59835	0	0	0	0.16958	0

Figure 13: An example of an ecological network created to model an oyster reef habitat. Inputs and outputs to the network are supplied at the bottom left, and the edge weights for each compartment in the network are supplied at the bottom right. Image provided by EcoNet.

of interrelated nodes in a network. The two main obstacles to this include isolating subsets of nodes within a network and calculating the amount of flow that passes through those derived subsets.

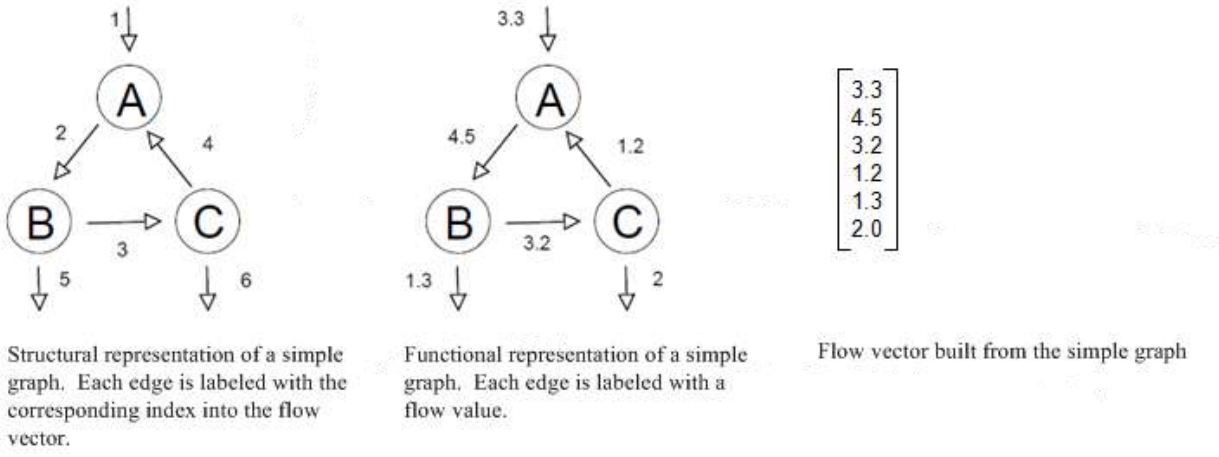
Graph Theory Constructs and Methodology

Graph Theory is a well-established field in mathematics and computer science. Describing a network as a graph allows expansion from a rich set of established constructs and methodologies. The following definitions are taken from Corman *et al.* [71]. A directed graph G is a pair (V, E) where V is a set of nodes or vertices and E is a set of binary relations on V called edges. A graph is connected if there exists a path from every node to every other node, and a directed graph is said to be weakly connected if replacing directed edges with undirected edges yields a connected graph. There are different ways of representing graphs, and this research uses the adjacency matrix construct. An adjacency matrix representation of graph $G(V,E)$ is a $|V| \times |V|$ matrix $A = (a_{ij})$ such that $a_{ij} = 1$ if (i, j) is an edge in G and $a_{ij} = 0$ otherwise. A path is a sequence of vertices or nodes of length k $< v_0, v_1, \ldots, v_k >$ originating at u and ending at u' such that $u = v_0$ and $u' = v_k$ and $(v_{i-1}, v_i) \in E$ for $i = 1, 2, 3, \ldots, k$. A simple path is a path where each vertex is unique. A cycle is a path where $v_0 = v_k$.

Network Decomposition

This research will pursue a computational algorithm to determine the partial throughflow for meaningful sub-networks in a network. First, flow vectors, sub-network vectors, and the sub-network matrix must be introduced. A flow vector can be constructed from a network if every edge in the network is labeled with a sequential integer and an edge's magnitude of flow is stored at the respective index (Figure 14). A sub-network vector (Figure 15) is a binary vector

with a size equal to the flow vector, where for any edge used in the sub-network there is a 1 at the corresponding vector index and all other elements are zero. A sub-network matrix (Figure 16) is a matrix of *n* rows and *m* columns where *n* equals the number of edges in the network and *m* equals the number of decomposed sub-networks. Each column of the sub-network matrix is a sub-network vector.

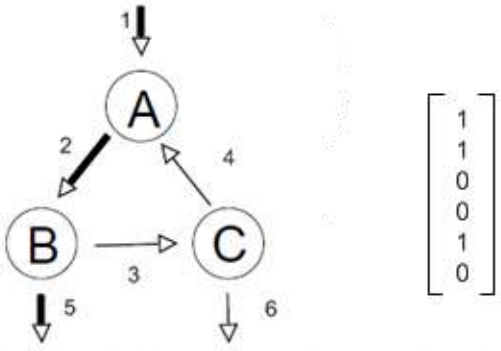


Structural representation of a simple graph. Each edge is labeled with the corresponding index into the flow vector.

Functional representation of a simple graph. Each edge is labeled with a flow value.

Flow vector built from the simple graph

Figure 14: The flow vector. Contributions for this image were made by Caner Kazanci and Rebecca Gaff.



Simple graph with a sub-network depicted with bold edges.

The sub-network vector made from the depicted sub-network.

Figure 15: The sub-network vector. Contributions for this image were made by Caner Kazanci and Rebecca Gaff.

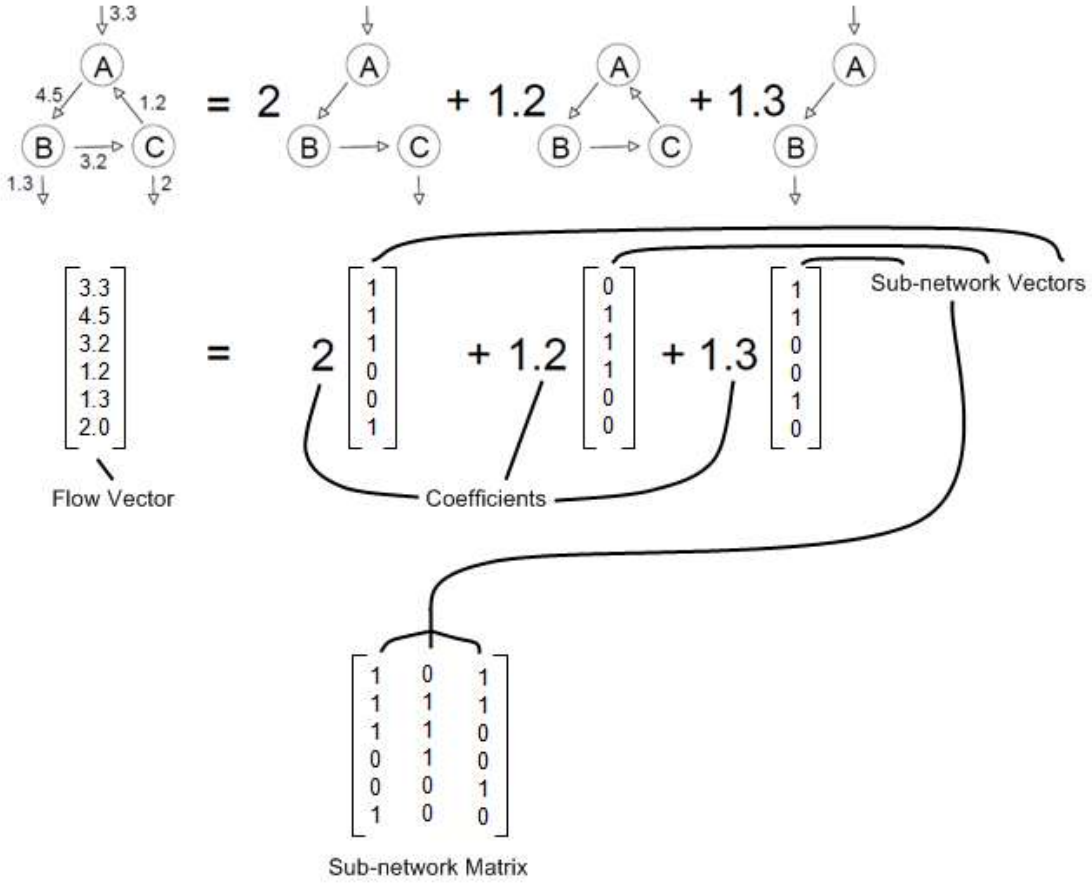


Figure 16: Example Decomposition. Contributions for this image were given by Caner Kazanci and Rebecca Gaff.

The definition of a relevant sub-network relies on understanding key concepts of decomposition and throughflow in a network.

- The network, and each sub-network, are assumed to be at steady state (input in equals input out; compartment storage is ignored).
- Edges in a structural decomposition are unweighted. The sub-network matrix is the output from a structural decomposition and flows across the edges are abstracted into coefficient terms pursued later.

- For a sub-network to be dissected from the rest of the network it must be self-sustaining, ensuring that any agent traversing a sub-network will remain in that sub-network without getting lost to some other sub-network. This effectively binds an agent solely to a particular sub-network.
- A constraint is placed on the decomposition of a network, that once decomposed the network (including flows) must be able to be recomposed. This constraint can be met by applying the derived coefficients (discussed later) to their respective columns in the sub-network matrix, and then summing the rows of the sub-network matrix. This produces the original network flow vector (Figure 16).
- Because a sub-network is unweighted and at steady state, each node in a sub-network must have only one input and one output. If nodes in a sub-network had multiple inputs or multiple outputs, an agent traversing the sub-network would have to choose which path to take and the sub-network could not remain unweighted.
- A sub-network in this methodology should be as small as possible ensuring flow coefficients computed later for each sub-network apply as specifically as possible.

Accounting for these concepts leaves a clear definition of a sub-network as any path through the network that starts where it ends, has no duplicate nodes, and where each node in the path has exactly one input and one output. This is the definition of a simple cycle.

It is acceptable in certain disciplines for a network to have inputs and outputs from an entity referred to generally as the "environment" (an entity outside of the graph that the graph can receive input from and send output to). In this paper the environment is considered to be another compartment in a self-contained network with no inputs or outputs.

A structural decomposition algorithm will now be outlined. Let *M* be an adjacency matrix for a network. If there are inputs to or outputs from the network, an environment compartment must be added to the network, and its edges must be listed in *M*. The decomposition algorithm takes *M* as input and places every compartment into a path of length 1. All the paths are placed into a queue. Until the queue is empty, path *p* is removed from the queue and subsequently inspected to see if it is a simple cycle of length greater than one. If the *p* meets this criterion, it is output as a sub-network. After inspection, paths of length *len(p) + 1* are constructed using every edge stored in *M* for the last node in *p*. Any new simple path (one that has not been created prior to this) is placed on the end of the queue and the loop continues.

```
Decomposition(AdjacencyMatrix M)

foreach node in graph
       path.add(compartment)
       queue.enqueue(path)

while (queue is not empty)
       path = queue.dequeue
       if (path is simple cycle)
              output simple cycle
       foreach edge in M
              for last node in path
                     queue.enqueue(path + new edge)
```

Figure 17: Decomposition pseudo code.

Uniqueness for Coefficient Vectors

Even in simple real world networks there is usually not a unique solution for computing the coefficients for each sub-network of a decomposed network. This is because different pathways through a network can be interpreted using different sets of sub-networks. Interpreting a pathway is building a parse tree of cycles used to comprise it (viewing it as a combination of

sub-networks rather than as a combination of individual nodes). An example of a path with multiple interpretations is seen in Figure 18. An interpretation of a pathway can be represented in two ways, first in a time series where a sub-network's location in the construct determines when it occurred, and second in an interpretation vector. An interpretation vector is a vector where each index represents a particular sub-network and a value in the vector represents the number of times the sub-network was used in the interpretation. Since the coefficients being sought are a representation of how much flow can be attributed to a particular sub-network, multiple interpretations pose a uniqueness problem. If a pathway can be interpreted using different sub-networks then sub-networks can be viewed as responsible for more or less flow depending on the interpretation assigned. This means multiple correct coefficient vectors exist, and they constitute a solution space of possible coefficient vectors.

This research presents a computational framework that analyzes simulated data presented from a network model and computes a coefficient vector for the specified network using a technique called functional decomposition. It requires as input a distribution of pathway data and an adjacency matrix for a network. The output is a coefficient vector. After a network is structurally decomposed, simulated data from the network can be analyzed to build an interpretation vector for each of the data instances in the distribution. Each pathway is interpreted as a combination of certain cycles from the structural decomposition and a frequency vector is used to keep track of how many times each sub-network is used throughout the entire distribution of data. To deal with the ambiguity that occurs when a single pathway can be generated by more than one combination of cycles, an averaging technique is introduced. This technique sums every occurrence of a cycle in all of the interpretations for a pathway and then adds a value to the frequency vector for that cycle equal to the sum of occurrences in all

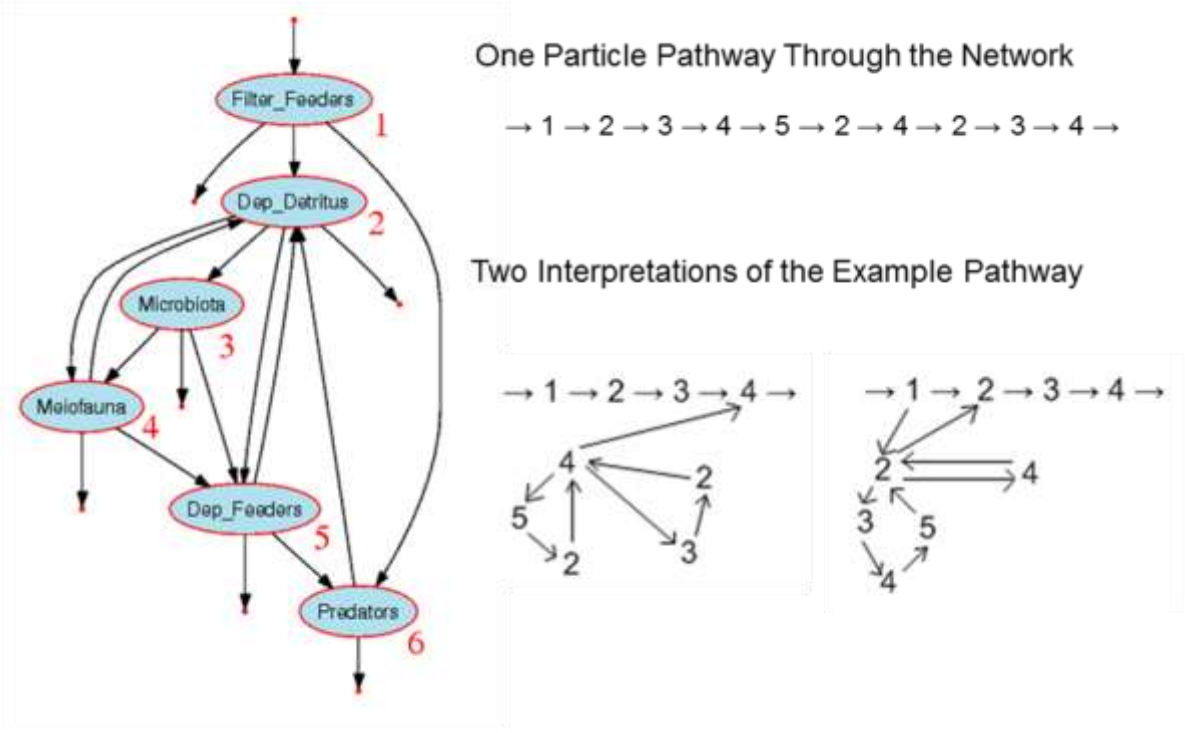


Figure 18: An ecological model of the oyster reef network is used to illustrate a path through a network that can be interpreted multiple ways (Interpretation 1 - cycle 1 [1 2 3 4] : cycle 2 [4 5 2] : cycle 3 [4 3 2]) (Interpretation 2 - cycle 1 [1 2 3 4] : cycle 4 [2 3 4 5] : cycle 5 [2 4]).

interpretations divided by the total number of interpretations. This technique aggregates multiple interpretations for a single pathway and averages their values giving equal weight to each of them.

The interpretation method currently used to parse a network pathway is an exhaustive search throughout all the possible combinations of cycles presented in the sub-network matrix. It is thought that this can be enhanced in two ways. One way to improve the efficiency of this process is to view interpreting a pathway through the network as parsing a string from a context-free language. The exhaustive set of simple cycles in the network can be used to build a context-free grammar covering all the possible pathways generated from a network. Once a grammar is

built the Cocke-Younger-Kasami (CYK) algorithm [74] (a dynamic programming algorithm) can interpret the string with a runtime of *O(n3)* where *n* is the length of the string being parsed. A second way of enhancing the efficiency of the proposed methodology would be interpreting data instances from a distribution in parallel. This could yield a substantial performance increase as the interpretation of each data instance is independent of any other data in the distribution, and could thus be interpreted disjointly. These enhancement suggestions are covered in the proceeding sections.

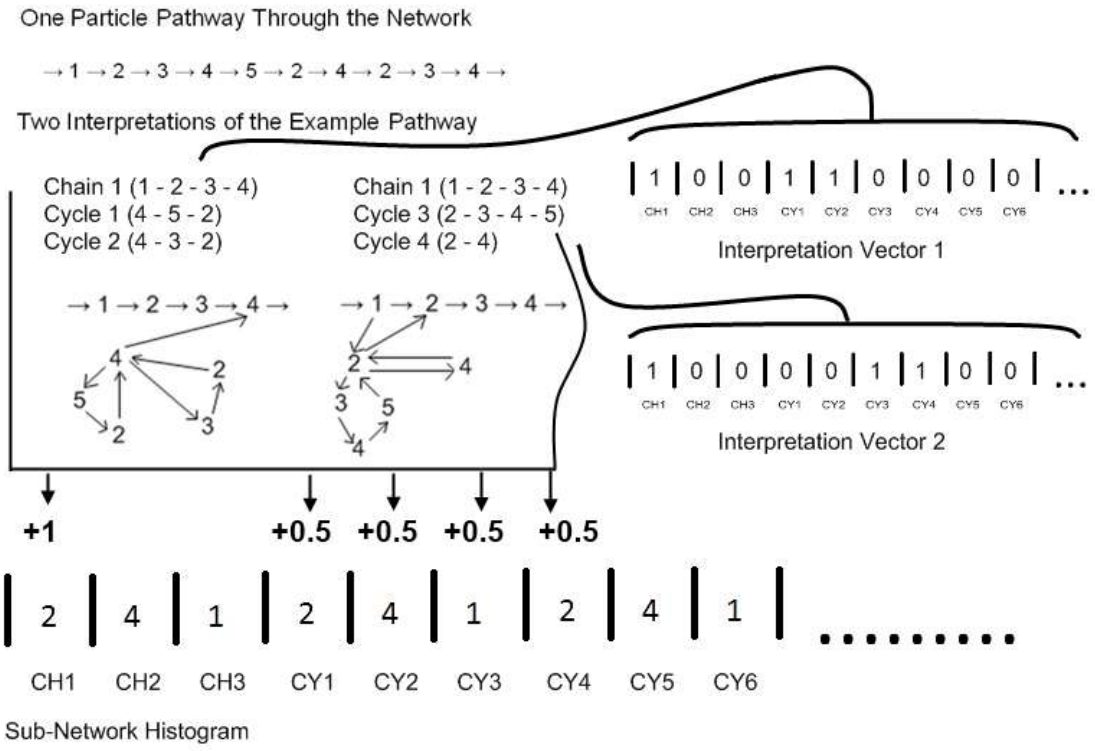


Figure 19: An overview of how the coefficient vector is found.

Data Mining with the Proposed Methodology

Decomposing a network and computing coefficients is a novel way to analyze networks, but much work is needed to determine useful ways of applying this methodology. Using the ascertained information for data mining holds potential to be transformative and is an interesting way of allowing machines to understand networks at a higher level of abstraction. One interesting data mining application could be to obtain a coefficient vector, zero out one or more of the coefficients, and then reverse the computational process to obtain a modified flow vector. The destructive effects of removing a particular group of sub-networks could thus be analyzed (*i.e.* how different is this modified flow vector if a particular group of sub-networks get removed along with all the flow attributed to those sub-networks). Other uses for this research cover a multitude of disciplines such as bioinformatics where network analysis is being used by Jiang *et al.* [72] for motif detection, network security where Ammann *et al.* [73] used network analysis to study network vulnerability, social network analysis where Carley *et al.* [74] used network analysis to study dynamic social networks (*i.e.* terrorist networks), and so many more fields where graph models can be built to study complex systems. The elegance of this methodology comes to life as the generality of it is discussed. This method can be applied to any problem that can be transformed into a directed weighted graph. A particularly interesting use for this approach would be symbolic time series analysis. Symbolic time series can be represented as a graph where an edge exists between two symbols if they appear next to each other in a time series instance. The weight for each edge is the number of times that particular edge occurs. If formulated this way a distribution of symbolic time series data could be analyzed using the proposed methodology and mined for valuable information. Symbolic time series can be used to represent a wide range of problem spaces, including those covered by numeric time series

analysis via some kind of symbolic transformation [75]. This technique could be viewed as a sort of unifying solution to a diverse problem set, and could be readily applied to a wide range of problems including music recommendation, economic network analysis, or social network monitoring.

Conclusion

This work has shown a methodology for decomposing a network into unique sub-networks and computing a coefficient vector detailing the fraction of total system throughflow for which each sub-network is responsible. An averaging technique was introduced to accommodate the ambiguity inherent in interpreting pathways through a network. Data mining opportunities for this methodology were postulated, but further research needs to be invested into exploiting the information present in the coefficient vector. The diverse range of problems to which this methodology could be applied was also discussed and is considered a major strength of the work.

Flow Decomposition in Complex Systems

Complex systems can be represented as weighted digraphs. Simple cycles play an important role in complex systems because they define the smallest unique groupings of nodes in the system. A grouping of connected nodes contains rich contextual meaning because of the relationships defined by its connecting edges. The work herein outlines a computational methodology to decompose the total throughflow of a complex system into a set of coefficients over its exhaustive set of simple cycles. A coefficient is computed for each simple cycle representing the amount of total system through flow the cycle is responsible for. This vector of coefficients provides information that can be used in data mining and information clustering applications to analyze the system. The proposed methodology provides a powerful framework

for analyzing symbolic data by assigning magnitude values to the contextual meaning within groupings of symbols.

Introduction

Systems analysis historically involves the evaluation of graph structure and function through the calculation of such metrics as nodal connectedness or compartment and through flows. This approach can be helpful, but lacks the ability to analyze interaction between groupings of connected nodes. Perhaps a more complete method of analysis includes a node's sphere of influence within well-defined node groupings (cycles). This approach can serve to contextualize the behavior of a node and provides a more complete understanding of its role to the graph.

The goal of this research is to quantify a measure of flow over groupings of interrelated nodes in a graph. It will be assumed that the set of simple cycles in the graph has already been found. For information on structurally decomposing a system refer to the previous section of this chapter. The work herein shows how a distribution of pathways through a graph can be interpreted using a context-free grammar derived from the exhaustive set of simple cycles in the graph. Interpreting a graph pathway in this manner allows each time step t in a pathway to be labeled as part of a particular cycle, and a vector can be built to represent the number of times each cycle has appeared in a distribution of pathways. This vector can be used to determine the flow magnitude for each cycle.

The structure of this paper is as follows. First, some basic concepts in formal languages will be reviewed. Then data mining on complex system will be discussed in conjunction with the computational techniques employed in computing the flow magnitude values. Finally, the

limitations of this methodology and some preliminary results will be presented together with discussion.

Context-Free Grammars

Sipser [24] describes a context-free grammar as a collection of substitution rules governing the definition of a regular language. These substitution rules allow grammatical units (groupings of literals) to be nested arbitrarily deep but not to overlap. Formally defined, a context-free grammar is a 4 tuple $G = (V, \sum, R, S)$ where V is a finite set of variables, $\sum$ is a finite set of terminal symbols, R is a list of substitution rules, and S is a start symbol. Starting from S, substitution rules can be applied to produce a parse tree where the leaves of the tree are terminal symbols and the internal nodes of the tree are variables. A parse tree holds the information on which grammatical units comprise a particular string. There can be multiple parse trees for a given string due to ambiguity in a grammar. In certain grammars it is not possible to eliminate ambiguity, and these grammars are called inherently ambiguous. When arranging a context-free grammar it can be necessary to formulate substitution rules in a simplified way called Chomsky Normal Form (CNF). A grammar is said to be in CNF if every substitution rule is either of the form $A \rightarrow BC$ or $A \rightarrow a$, where A, B, and C are any variables, except B and C are not the start state, and a is any terminal.

Data Mining on Complex Systems

A system is a grouping of interconnected components that form a whole entity. Figure 20 depicts an example of an ecological system created to model an oyster reef habitat. A complex system is more a notion of intuition than definable characteristics due to the fact that there is no agreed upon definition of system complexity. In the absence of an agreed upon general theory, one is left describing traits that are present when a system is thought to be complex. A complex

system is typically comprised of building blocks (*i.e.* atoms, molecules, cells, *etc.*; more generally agents), which can be complex systems themselves (giving rise to hierarchical levels of interaction). These building blocks interact with each other to exude actions, traits, behaviors, and organization that are not observable from studying the building blocks in isolation [19]. Interaction between compartments is typically illustrated through edges in a weighted directed graph. Extending these relationships beyond a single edge defines relationships with greater contextual meaning. The set of all simple cycles in the system defines an exhaustive set of unique, self-sustaining relationships over that system. These relationships can support themselves, existing in isolation. As a unit of flow moves through the system, any pathway it takes can be described as some combination of these defined relationships. This is a valuable source of information about the system because the frequency of occurrence for these unique relationships within the system can be tracked.

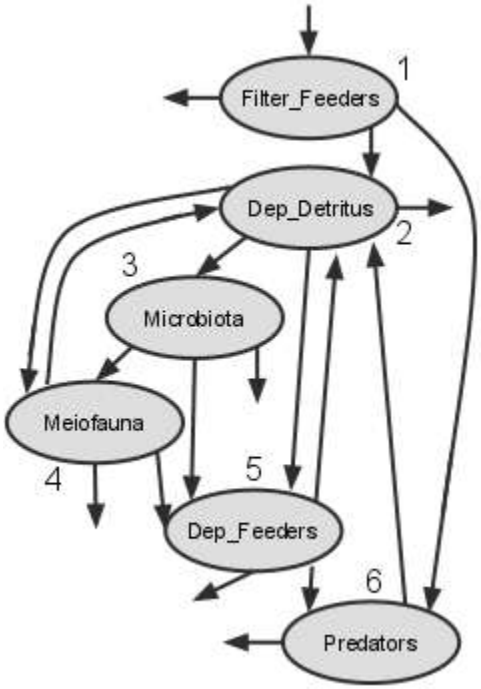


Figure 20: A model of an oyster reef ecosystem.

To calculate magnitude values (frequency of occurrence) for each simple cycle in a graph, a data simulation technique called Network Particle Tracking is used to simulate network activity over a specified range of time in the networks life. The data Network Particle Tracking creates is a list of pathways that particles traversed through a system. Each of these pathways will be treated as a string generated from a context-free grammar defined by the set of cycles within the system. In this grammar each unique cycle within the system is a grammatical unit within the context-free language of the decomposed graph. Each pathway in the data distribution is parsed to find the grammatical units comprising it, and a vector is kept representing each time the different cycles have been used. This vector contains the magnitude values.

Constructing a context-free grammar from the set of cycles in a graph is central to the proposed methodology. A context-free grammar is a 4 tuple, as defined earlier, $G = (V, \sum, R, S)$. The variables (V) for a grammar in the proposed methodology are all compartments in the graph. It is important to note that in systems research there is an idea of an environment that inputs into the system and receives output from the system. The environment can be added to the adjacency matrix as a compartment and is treated as a special node in the graph where pathways must start and end. The terminal symbols in the grammar $(\sum)$ are also all compartments in the graph. The substitution rules for a grammar (R) are obtained from the cycles in a graph. Any variable in the grammar can be replaced by any grammatical unit where that variable is the first element in the unit, or by its corresponding literal.

Two distinct classes of cycles exist in a grammar, those that contain the environment and those that do not. Any pathway through the system can be interpreted using exactly one cycle containing the environment (as paths must start and stop in this node) and zero or more cycles

```
          Variable:  X
  Grammatical Unit:  X Y Z
Substitution Rule:  X → X Y Z X
```

Figure 21: Example of a substitution in a context-free grammar.

that do not contain the environment (internal cycling loops). Each cycle not containing the environment represents multiple grammatical units because there are multiple orders in which the compartments of the cycle can appear. An internal cycle represents a number of grammatical units equal to the length of the cycle. These different grammatical units are made by assigning a new index to a node in the cycle equal to its current index plus one mod then length of the cycle. This is process applied to find every possible ordering of cycle compartments for every internal cycle. The pseudo code for this can be seen in Figure 22. Each cycle containing the environment represents a single grammatical unit because the environment is the starting and stopping point for each pathway, thus determining the order the nodes for these cycles must

```
foreach internal cycle:
   for i in range(len(cycle)):
      new cycle[len(cycle)];
      for j in range(1,len(cycle)+1):
         x = j%len(cycle);
         new cycle[x] = cycle[j-1];
      cycle = new cycle;
```

Figure 22: Pseudo code for finding all grammar units within a cycle.

appear in when being used to construct a pathway. Finally, the environment is designated as the start symbol *(S)*.

Once the grammar is built it is possible to parse a pathway through a graph to calculate the grammatical units (cycles) the pathway traversed. To compute a parse tree the CYK algorithm is used. The CYK algorithm is a dynamic programming algorithm that considers every possible subsequence of the sequence of words presented in a string, and compares these subsequences against the grammar to see if they are interpretable. The results are stored in a table that can be traced back through to construct all parse trees associated with the input string. The runtime for this algorithm is $O(n3 * |G|)$ where n is the length of the input string and $|G|$ is the size of the grammar. Grammars presented to this algorithm need to be structured in a simplified way called Chomsky Normal Form (CNF). As mentioned earlier, a grammar is said to be in CNF if every substitution rule is either of the form $A \rightarrow BC$ or $A \rightarrow a$, where A, B, and C are any variables, except B and C are not the start state, and a is any terminal. Any context-free grammar can be transformed into a CNF grammar expressing the same language [24]. This transformation can lead to considerable bloat in the size of the grammar. In the worst case this can increase grammar size from g^2 to 2^{2g}. For implementation of the proposed methodology, each rule in the original grammar requires a number of rules in the CNF grammar equal to one minus the length of the right hand side of the rule. The transformation is achieved by introducing new symbols into the grammar. A rule in the original context-free grammar representing a cycle *ABC* would appear as follows $A \rightarrow ABCA$. This rule is transformed into three rules $A \rightarrow AX$, $X \rightarrow BY$, $Y \rightarrow CA$ representing the same cycle where X and Y are symbols not previously in the grammar. The number of rules to be expected would be equivalent to $\sum_{i=0}^{n} len(cycle[n]) - 1$ where n is the number of cycles in the set of cycles over the graph. Once

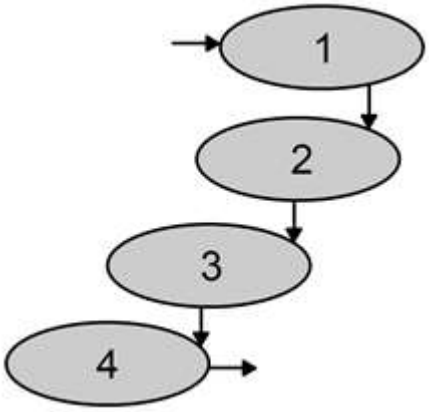
Pathway through system illustrated in Figure 20

E → 1 → 2 → 3 → 4 → 5 → 2 → 4 → 2 → 3 → 4 → E

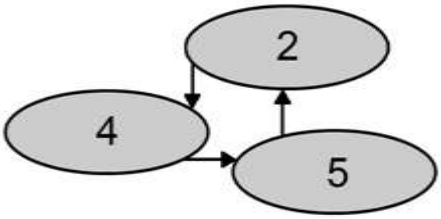
Parse generated for this pathway

E	1	2	3	4	5	2	4	2	3	4	E
A	A	A	A	B	B	B	C	C	C	A	A

Cycle A : E 1 2 3 4 E



Cycle B : 4 5 2



Cycle C : 4 2 3

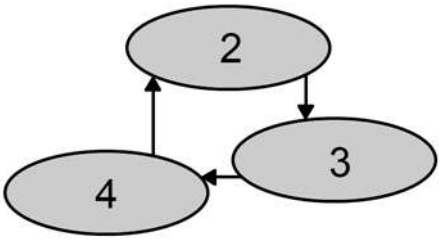


Figure 23: A parse of an example of a pathway through the system in Figure 20.

the grammar is transformed the CYK algorithm is used to analyze a distribution of pathways simulated over the complex system by Network Particle Tracking. The distribution of pathways can be of arbitrary size, but the accuracy of the magnitude values increases as the number of analyzed pathways increases. To calculate the magnitude values a vector is kept over the entire distribution of data, tracking the occurrences of each cycle.

Discussion

The vector calculated by the proposed methodology represents the occurrence frequency of each cycle in the exhaustive set of simple cycles. The occurrence frequency is the number of times a unit of flow passed completely through a cycle grouping. This methodology takes symbolic sequences of data produced from a model and computes numeric magnitude coefficients reflecting some position within an *n* dimensional feature space. This can be seen as a symbolic transformation routine that would provide valuable results for machine learning and computational intelligence methodologies. The frequency vector allows insight into the position of a model within a feature space, thus offering information pertaining to what state a model was in when the data being analyzed was produced. These coefficients could also be used as a distance metric to calculate the difference between two distributions of data from the same model. Furthermore, it would be possible to compare specific coefficients, or groupings of coefficients, between two models, allowing one to search for relationships that most differentiate the two data distributions.

The ambiguity within the grammars produced by this methodology has not been discussed until now. Ambiguity in a grammar means that a particular input string can be legally generated by multiple parse trees. As it pertains to this work, it is possible to interpret some pathways through graphs using different sets of cycles. The implications of this ambiguity must

not be ignored if a correct vector of magnitude values is to be calculated. There are two different ways to deal with these multiple interpretations when they arise. The first way is to find all interpretations of a pathway, and then for each cycle add to the frequency vector a value equal to the total number of occurrences of that cycle, in all interpretations, divided by the total number of interpretations. This method uses each of the interpretations and assigns a weighting parameter to them. The resulting vector from this method is interesting because when multiple interpretations happen they are all utilized, and their combined occurrence if reflected. This results in a vector of averaged values as they occurred in simulation. This method also reflects the probability of ambiguous cycle combinations occurring in the same pathway, which is defined by the systems flow parameters. The second method of dealing with ambiguity is to find all interpretations of a pathway and maintain another computationally derived entity called a Difference Vector Matrix (DVM). Every interpretation of a pathway is a vector where each index in the vector represents a cycle, and the value at the index represents the number of times the cycle was used in the interpretation of the pathway. When multiple interpretations happen, these localized interpretation vectors can be subtracted from each other to produce a Difference Vector that encodes the ambiguity. These Difference Vectors define equality constraints between groupings of vectors. In this method only one interpretation is added to the frequency vector but these Difference Vectors are tracked and maintained in the DVM. A Difference Vector is added to the DVM if it increases the rank of the DVM. This additional matrix retains the information needed to bound the solution space of possible frequency vectors calculated by this methodology. This is a novel way to isolate ambiguities in context-free grammars.

The proposed methodology has runtime constraints that must be addressed. The runtime is bounded by the runtime of the CYK algorithm used for interpreting the cycles appearing in a

given pathway. The runtime of the CYK algorithm is $O(n3 * |G|)$ where n is the length of the input string and $|G|$ is the size of the grammar. The most significant factor in this runtime is the length of the input string. This means that graphs producing shorter pathways from input to output require less computation. As the internal cycling within a graph increases the runtime increases exponentially. The size of the grammar also factors into the runtime to a lesser degree. The size of the grammar depends both on the size and connectedness of the graph. Methods of dealing with the runtime constraints are currently being researched. It is thought that through strategic graph contraction and distributed computing this methodology can overcome these constraints to prove useful even for large, heavily connected graphs with large amounts of internal cycling.

For testing purposes this methodology was run on three ecological models of varying size and complexity. The models were the oyster reef model, the Georgia salt marsh model, and the Neuse river basin model. The oyster reef and the Georgia salt marsh models were able to be run fully and the magnitude coefficients for each of the cycles in the graphs were computed. The Neuse river model posed a greater challenge do to the size and connectedness of the model, in combination with its heavy proclivity for internal cycling. Pathways for this model routinely had lengths of greater than 40 nodes with the occasional pathway reaching lengths of 100 to 200 nodes. Memory issues were the main problem with this model as the current implementation of this methodology uses RAM memory and not disk memory while computing the CYK algorithm. If disk memory were to be used the memory constraint could be overcome, but this would still leave a substantial runtime to deal with. As aforementioned, research is being directed into dealing with these problems.

Conclusion

The research herein has shown that a weighted directed graph can be decomposed both in structure and function to produce a vector of magnitude values for each of the unique relationships within the graph. The importance of these magnitude values has been discussed along with the current limitations of this methodology.

CHAPTER 4

APPLICATION

Network Flux Analysis in the Neuse River Estuary

Complex systems can be represented as weighted directional graphs. Fluxes (sequences of directed arcs) play an important role in digraphs because they define relationships consisting of unique groupings of compartments. A grouping of connected compartments contains rich contextual meaning because of the relationships defined by its connecting arcs. The flow activity over these contextually rich compartmental groupings, called fluxes, holds valuable information about the state of a model. Functional decomposition of system throughflow transforms a system model into a vector within a numeric feature space well suited for system state classification and data mining. In this paper an ecological model with four years of seasonal data is analyzed. First, coefficients are calculated over the cycles within the system, and then data mining using feature selection is performed on the coefficient vectors to isolate coefficients that optimally differentiate seasonal variance. A Support Vector Machine (SVM) will then be used to show the validity of the isolated features for classifying the seasonal models as belonging to a particular season. Finally, how the methodology might be used to expose hidden relationships in complex system networks is discussed.

Introduction

Complex systems can be modeled as a network of interconnected compartments with directed, weighted arcs (currency flows; *e.g.*, mass or energy). This type of directed graph (digraph) construct appears across a wide variety of scientific disciplines [73, 76, 77, 78, 79, 89]

including economics, computer science, ecology, biology and sociology. The models can be structurally and functionally decomposed to produce a set of fundamental relationships (fluxes) within the system and coefficient values representative of the total system throughflow accounted for by each of the relationships [87]. Coefficient values for each flux represent the units of currency passing through the graph segment in simulated or real-world data. System traits can be encoded in the system using sets of fluxes, and the expression traits is encoded by the coefficient values over the fluxes in the set.

In this paper Network Flux Analysis (NFA) is used to analyze the system trait of seasonality. NFA provides the framework for analyzing the building blocks of complex systems as they relate to higher level system function. It allows systems to be studied through hierarchal layers of fundamental compositional units.

The most basic unit of a system is a compartment. Compartments are unique agents within a system where network flow resides and is potentially exchanged with other compartments. A compartment affects other compartments in a graph either directly or indirectly. Compartments can be grouped into self-sustaining, fundamental sub-networks within the system called fluxes. A flux is defined as a directed grouping of compartments from a system meeting the following constraints:

- Compartments n and n + 1 in the flux share a directed edge in the system from n to n + 1.
- A flux must end where it begins; the first compartment must also be the last compartment.

- A compartment can appear at most one time in a flux, except for the first compartment which must appear exactly twice; the second occurrence being the last compartment in the flux.
- The environment is considered a compartment and can appear in a flux only as the first and last compartment.

Fluxes are the smallest steady-state groupings of unique compartments in the system, and are important compositional elements of a system. Fluxes can be grouped together into diagnostic sets whose flow activities encode system traits at particular values (*e.g.* the system trait season has four values: spring, summer, fall and winter). NFA, as seen through this perspective, presents a construct for complex systems analysis reminiscent of how genetics helps understand biological organisms. In this analogy compartments become the nucleotides of complex systems, fluxes are codons, groupings of fluxes can be seen as genes, and the coefficient values over a grouping of fluxes can be seen as gene expression (the particular value of a system property encoded in the flux set).

NFA uses the coefficient values for fluxes as numeric inputs into a machine learning algorithm. This allows for data mining on systems to isolate a minimal set of fluxes responsible for encoding system traits. In this paper the flux set being sought is one whose values predictably fluctuate between seasons. In NFA, each flux's value represents a variable (degree of freedom) in a numeric space and machine learning algorithms can analyze this numeric space to extract information. The values of the flows will change seasonally, and quantification of this imparts a particular seasonal signature to the system. A subset of fluxes that encode a trait is

valuable information giving domain analysis the ability to infer system status and isolate flux indicators based on prior data.

The structure of this paper is as follows. First, functional decomposition and machine learning (specifically feature selection and Support Vector Machines) will be reviewed. Then, the specific machine learning algorithm of this paper will be described and illustrated by an example ecological model from the Neuse River estuary, North Carolina [101, 102, 103]. Finally, the results will be interpreted and strengths and limitations of the methodology will be discussed.

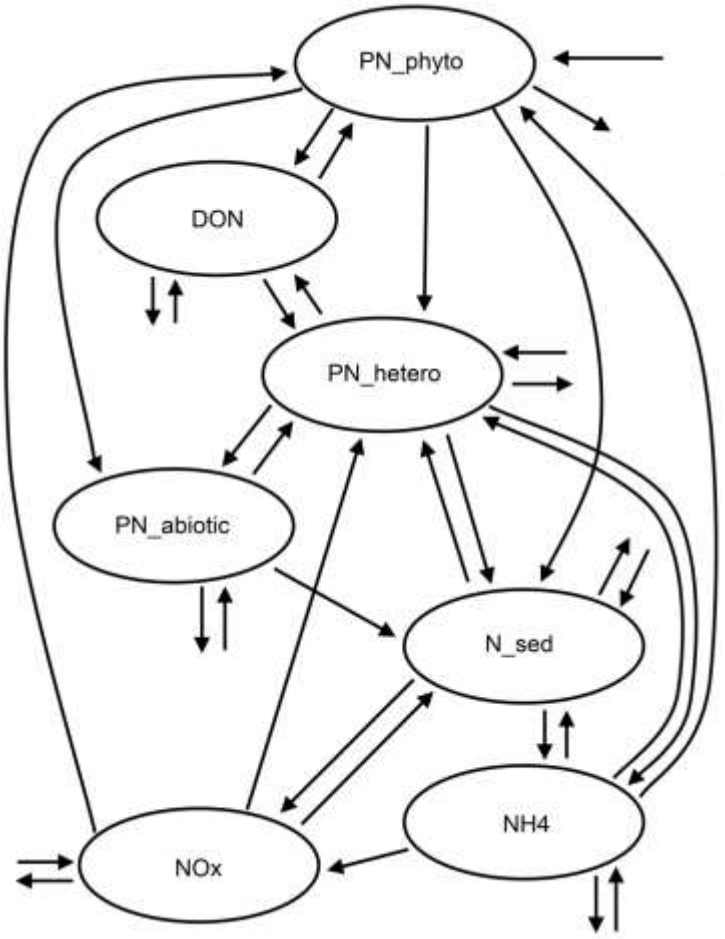


Figure 24: EcoNet model of the Neuse River estuary.

Machine Learning

Supervised machine learning algorithms can be used to approximate linear and non-linear functions for data classification. These algorithms build learning models from training data, and they use these models to classify new data where the class is not known. The training data contains rows data of data with *n* columns (features) from an *n* dimensional numeric space *S*, and a label telling the algorithm which class the data row belongs to. The training data describes locations in the numeric space for each class of data, and they are treated as a representation of a non-linear function *f(i)*, where *i* is a row in the training data. Once the training data are defined, a model is constructed to fit it.

While constructing the model a portion of the training data is placed into another grouping of data called the validation data. The validation data are withheld from the learner while training it, and used as a final test to evaluate how effective the model generalizes beyond the training data. There are different schemes for defining the validation data. This research uses a method called *n* fold cross validation. In this technique the training data are divided into *n* data groups and *n* learning models are built such that each of the data groups is used as validation data for one of the models. The *n* models are then combined to produce a single model that is used for classification. This technique helps the model generalize better when the amount of training data is small.

Feature Selection

In machine learning a feature is variable input into a learning algorithm. If data are put into rows, features are the columns that intersect each row. Feature selection is the process of isolating a subset of important features that perform optimally when used in learning algorithms. When data are sent through a learning methodology like a Support Vector Machine, only

selected features are used for data classification. In machine learning the dimensionality of the numeric feature space is defined by the number of features (columns) from the data supplied to the learner. As this feature space grows in dimensionality, it has been shown [98] that algorithms have a tendency to over fit to the data they are being trained on, resulting in poor generalization to unknown data. Another problem with large feature spaces is that as the number of features grows, learning results can degrade because of the likelihood that features irrelevant to the concept being learned (*i.e.* seasonal classification) are included for evaluation by the algorithm. These irrelevant features act as noise that confuses the algorithm and degrades results.

Two distinct types of feature selection methodologies are subset selection and variable ranking. In variable ranking some metric is posed against all features and a score is produced. The score of each of the features is indicative of its performance for the input being evaluated. In subset selection, groupings of features are evaluated to see how they perform together at the desired learning task. A feature subset α (of variable size) is chosen, and the training data are sent through a learning algorithm (*i.e.* a Support Vector Machine) for classification using only the features in α. The classification results are used to grade how well α can classify the data. Since exhaustively searching all combinations of features is impractical except for small sets of features, heuristic search strategies are used to explore combinations of features in a calculated manner.

Support Vector Machines

Support Vector Machines (SVM) learn concepts by separating data into two classes and treating them as two generalized sets of vectors in a numeric space. The SVM finds a separating hyperplane between the two classes of points in the numeric space which is the maximum

distance from either of the two (Figure 25). SVM's have been shown to outperform other supervised machine learning algorithms over a wide variety of problems. This is because they generalize better, due to the nature of how they learn. Other machine learning algorithms can find hyperplanes that separate data sets, but the SVM finds the hyperplane with the greatest distance between either of the two classes. The SVM finds support vectors, which are data instances from either class that are the closest to the opposite class in the numeric space being analyzed. Once these support vectors are found, geometric operations are applied to find the hyperplane that is maximally distant from both sets of support vectors. Finding support vectors

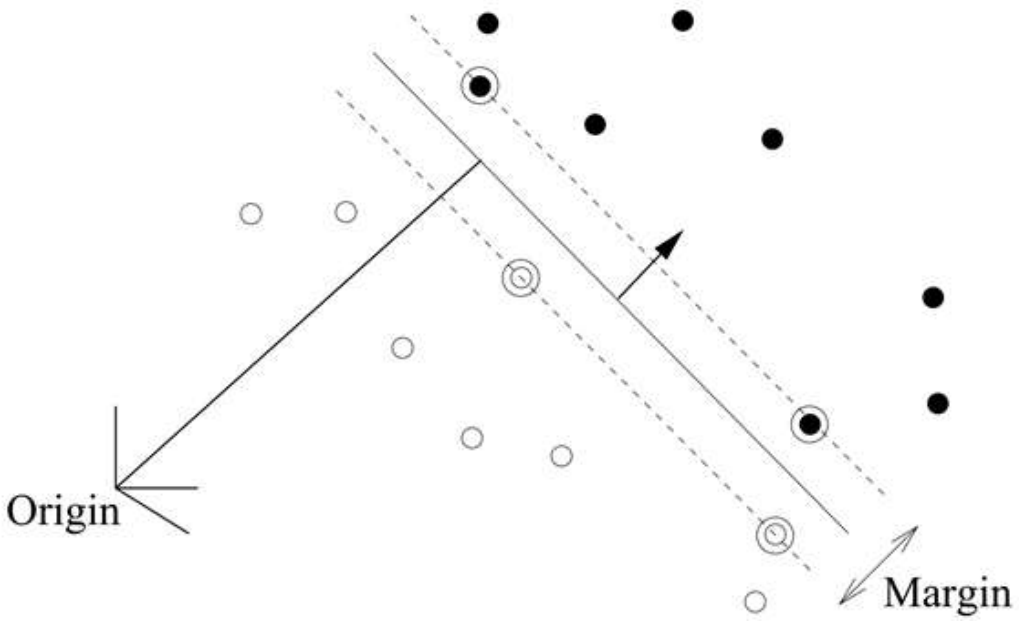


Figure 25: Illustration of what a maximum marginal hyperplane looks like between two sets of data instances in a feature space (image taken from Christopher J.C. Burges [47]). The black and white dots represent different classes of data located numerically in this two-dimensional feature space. The circled dots are support vectors because they are the dots closest to the opposite class of data. To generalize optimally the SVM will find a hyperplane that separates these support vectors by class and is maximally distant from both classes. The solid line in the middle of the two dashed lines is the hyperplane in this example. When given new "unknown" data the SVM would classify anything on the white dot side of the solid line as a white dot and anything on the black dot side of that line as a black dot.

and computing the maximum marginal hyperplane is a standard quadratic programming problem [47]. This explanation assumes a linearly separable feature space because it is the easiest way to explain the concept. SVM's can be generalized to support nonlinear features spaces as well as problems with more than two classes of data.

Flow Decomposition within Systems

Decomposing throughflow in a complex system allows for rich data analysis due to the fact that coefficient values for fluxes form a numeric feature space which can be leverage in many different kinds of machine learning and data mining methodologies. Luper *et al.* [87] outlined a computational methodology for functionally decomposing system throughflow, in a similar way to earlier work by Ulanowicz [104] in which he identified both acyclic and cycle paths within a system and listed their arc weights. The present methodology transforms a structurally-decomposed system into a context-free grammar and uses the Cocke-Younger-Kasami (CYK) [90, 91, 92] algorithm to calculate the flux coefficients derived from an individual-based simulation of the system. The CYK algorithm parses each individual path to label each step of the path as belonging to a specific flux. The frequency of occurrence for each flux is tracked throughout the data and stored in a vector. This vector is the basis for the coefficient values sought in the functional decomposition. The details of the methodology are outlined in the first author's previous work [87, 105].

Classification of the Neuse River Ecological Data

The Neuse River ecological model is a seven-compartment model that represents nitrogen flow in the Neuse River estuary in North Carolina. This is a heavily connected model with a high cycling index (0.89), meaning most of the nitrogen in the system cycles extensively between compartments during the time between entry and exit. The models analyzed in this

paper were created from seasonal nitrogen data obtained over a period of four years [101, 102, 103]. The seasonal data were derived from samples systematically collected in the field.

The methodology proposed by this analysis will seek to analyze functional decompositions of the Neuse River estuary to isolate fluxes important to seasonal variability. In order to create the data necessary for flow decomposition, a simulation technique called Network Particle Tracking (NPT) [99] is used. NPT is an individual-based simulation method, where discrete quanta (particles) of material or energy are labeled and tracked in time as they flow through the model compartments. NPT starts with breaking input flows into discrete packets which are called particles. For the present analysis a particle represents a nitrogen atom. Based on flow rates, NPT determines which flow is likely to occur and when. A particle is chosen randomly from a donor compartment and introduced to a recipient compartment. Ecosystem models are open systems and therefore new particles enter the system continuously. At steady state, when one of the particles enters a compartment or the system, another must leave. So if the chosen flow is an environment input, a new particle is labeled and introduced to the recipient compartment. NPT keeps the record of path history of all particles, including when and where each particle movement occurs. These data are dumped into a text file after the simulation ends. NPT is particularly useful because, unlike similar individual based algorithms, it deduces all the rules on individual particle movement from the flow, input and output rate coefficients of the model. Therefore no additional information is needed to run an NPT simulation. NPT is a stochastic method that is compatible with the differential equation representation. In other words, for the same model, the average of many NPT simulations agrees with the differential equation solution.

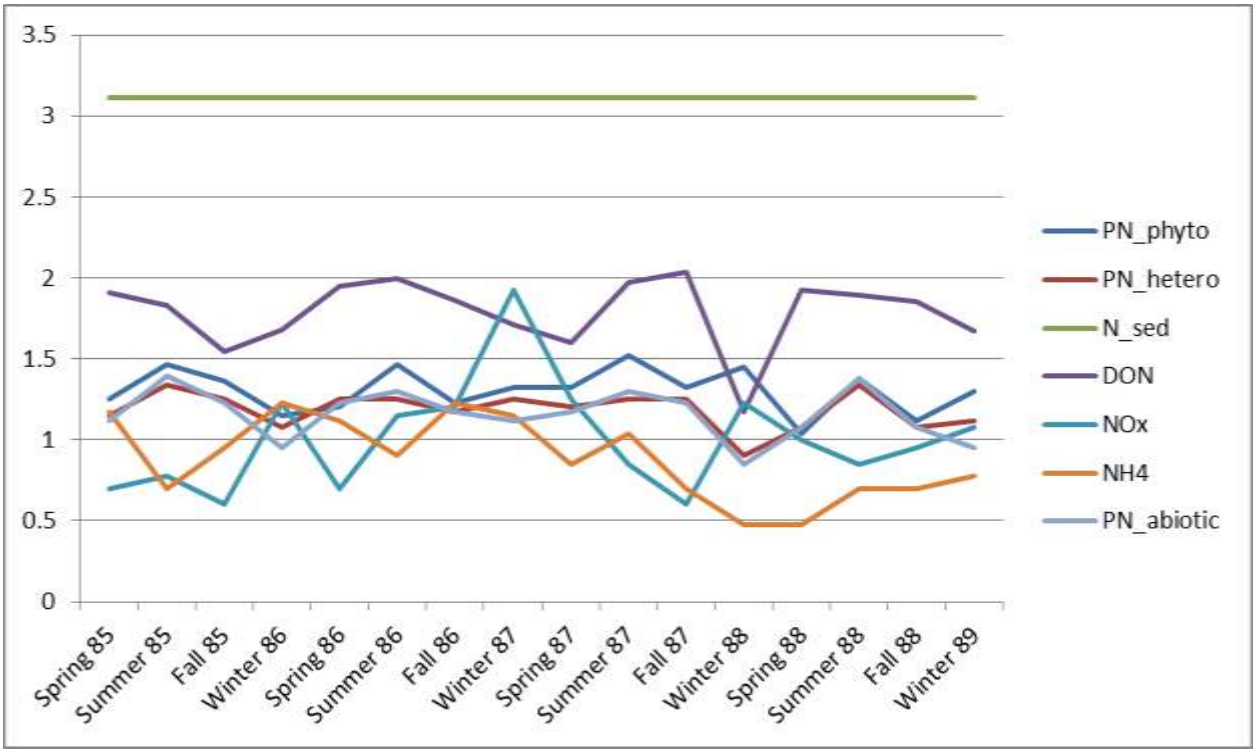


Figure 26: Storage values for each compartment in the Neuse River estuary in North Carolina plotted over time. Data on this graph were logarithmically scaled and derived from data collected in the field representing 16 discrete time periods (four per year) from 1985 through 1989.

For this analysis NPT was set to simulate around 10,000 paths per model (16 models). The accuracy of a functional decomposition increases as the number of analyzed paths increases; it would therefore be optimal to interpret all simulated paths for each seasonal data set, however the computational resources available for this work did not permit that. This analysis used different size constraints for the number of paths given to the functional decomposition routine allowing for the comparison of the results over varying degrees of functional decomposition accuracy. Six functional decompositions were performed on each of the 16 models with the number of paths analyzed set to 25, 50, 75, 125, 250 and 500. This process produced six sets of 16 functional decompositions where the functional decompositions in each set were derived from

a different number of randomly sampled paths from the 10,000 available paths for each model. Analysis was then performed on each of these six data sets as well as a seventh where the decompositions from the groups of sampling sizes 250 and 500 were lumped into a single dataset consisting of 32 functional decompositions. This process of composing six sets of functional decompositions was necessary because of limited computational resources. In a scenario with less computational constraints, all available paths could be included for each model, and only one set of 16 functional decompositions could be produced. Analyzing the data in this way has the benefit of allowing comparative analysis towards sampling sizes used in functional decomposition.

Once data sets are functionally decomposed into weighted simple cycles and chains we call fluxes (e. g., the Transformed Data column of Figure 27), feature selection is used to isolate subsets of these fluxes best suited to identify a specifiable system trait. In this case, to illustrate the methodology, the trait selected was seasonality. The problem then becomes this: What is the minimal flux set needed to specify the season from which the data instances in question are derived with maximal accuracy? Figure 28 shows a simulated depiction of how supervised machine learning classifies flux data as belonging to seasonal labels.

There are 640 unique fluxes in the Neuse River estuary model and the coefficient for each one is a feature (methodology input variable). The numbers of features (flux coefficients) in this model makes an exhaustive search over all combinations impossible; therefore this analysis uses standard heuristic approaches to search for good combinations of features. This analysis utilizes different search strategies working in a combined effort to extract optimal subsets of features (fluxes) from the large search space of possible combinations. The methodologies used were a combination of genetic algorithm [94], greedy stepwise search,

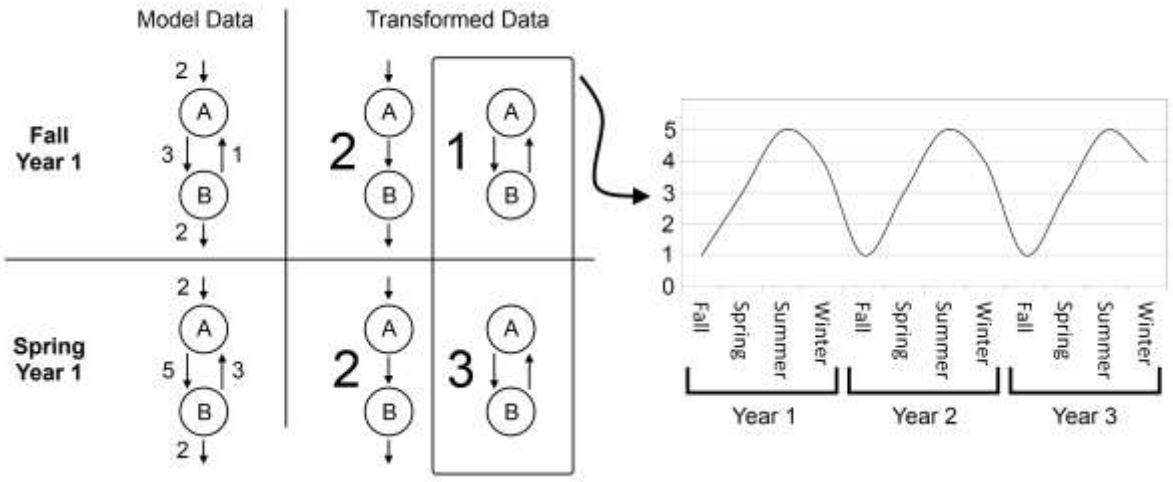


Figure 27: This figure illustrates two instances of a single structural model having different flow weights. Two different flow-weighted instances of the model are shown in the Model Data column. The Transformed Data column depicts the fluxes and their coefficient values after functional decomposition. The graph on the right illustrates the second column of coefficients in the Transformed Data column plotted over time where the vertical axis represent the coefficient value (the graph assumes there are 10 more flow weight model instances that are not graphically depicted in the figure).

forward selection [95], rank search [96] and scatter search [97]. Since an exhaustive search over all feature combinations is not possible any results from this methodology are not unique or guaranteed to be optimal.

The feature selection process was run numerous times producing different feature combinations that were then given a final evaluation. The final evaluation was a blind classification of the functional decompositions in an SVM using only the isolated features. The final understanding of how well a feature set describes seasonal variation is derived from the accuracy with which an SVM can blindly classify the 16 functional decompositions using only the feature set (fluxes) being evaluated. The groupings of fluxes encoding seasonal variation

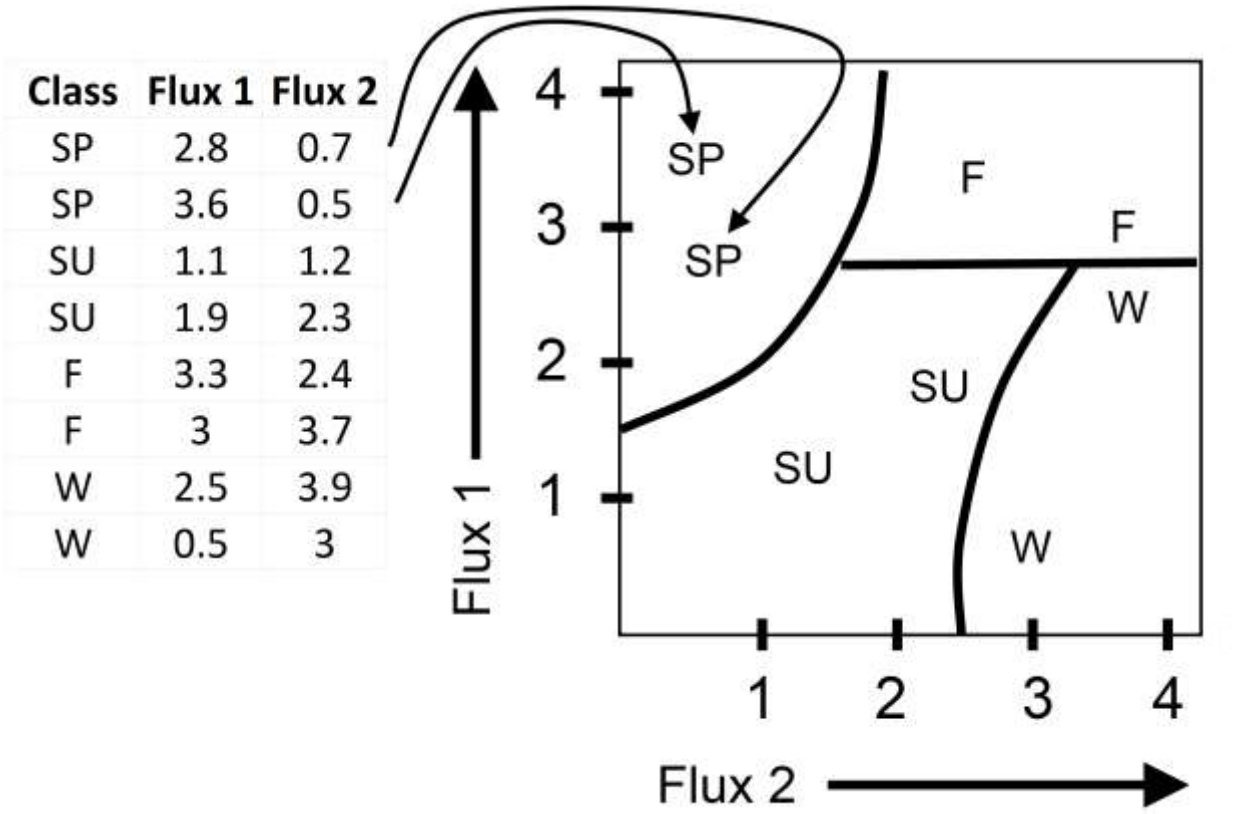


Figure 28: Given the above data distribution a two-dimensional plot (two-dimensional feature space) shows how the feature space comes to be understood by a learning algorithm. The lines in the two-dimensional plot separating the classes of data become conceptual boundary lines for the learning algorithm to understand the classification of the data instances. The data table presented here is data the algorithm will train on; the classes for the instances in the table are known apriori and given to the learning algorithm. The question now becomes the following: Given a new data instance whose classification is not known, what region of this two dimensional plot does the instance fall into? As far as the learning algorithm is concerned the region this data falls into is the classification it is statistically most likely to be. This is a simple problem in two dimensions with clear boundary lines. In a real world problem placing data instances into feature spaces can be a complex process. The data instances for hard problems are typically not linearly separable.

gives ecologists new information about the ecosystem that is important. A grouping of fluxes produced by this methodology represents a small diagnostic unit within the system that fluctuates in a predictable manner with respect to seasonal variability.

Results

Running the aforementioned methodology on the Neuse River estuary models isolated a set of eight fluxes (of the 640 total) that collectively allow an SVM to classify each of the 16 models seasonally with 100% accuracy. Another way to understand this is that accuracy of the classification results is greatest when only these eight fluxes are used and no others. It should be noted that this set performed significantly better than any other set found during this analysis, but is not guaranteed to be a unique or optimum solution. Other sets of fluxes could exist that vary seasonally in a predictable manner, producing similar results to the set of eight reported herein. The paths corresponding to the eight fluxes found by this analysis, as well as the composition of all eight into one sub-network are shown in the figures on the following pages.

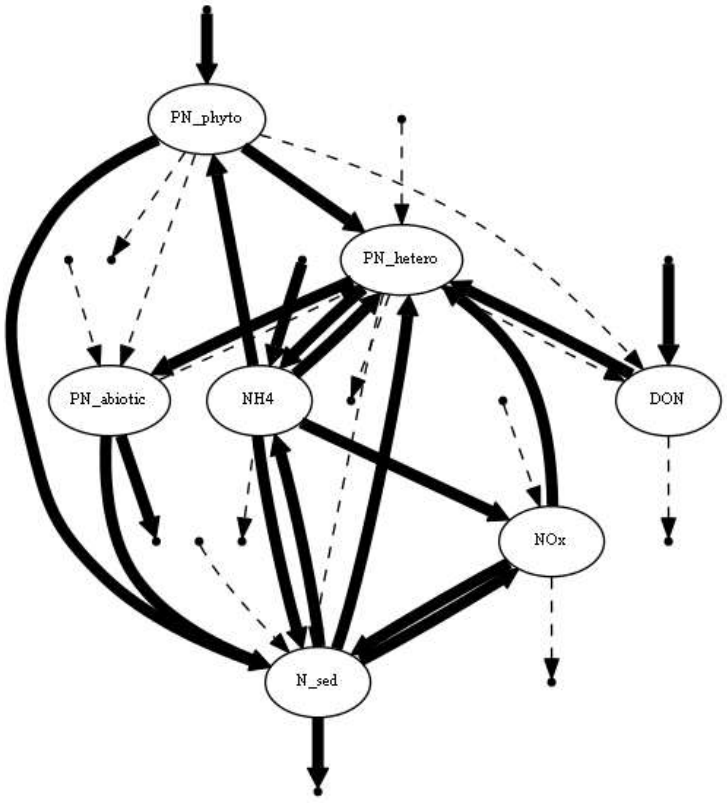


Figure 29: Graph of the Neuse River ecosystem model. All edges used in the eight fluxes found during feature selection are in bold.

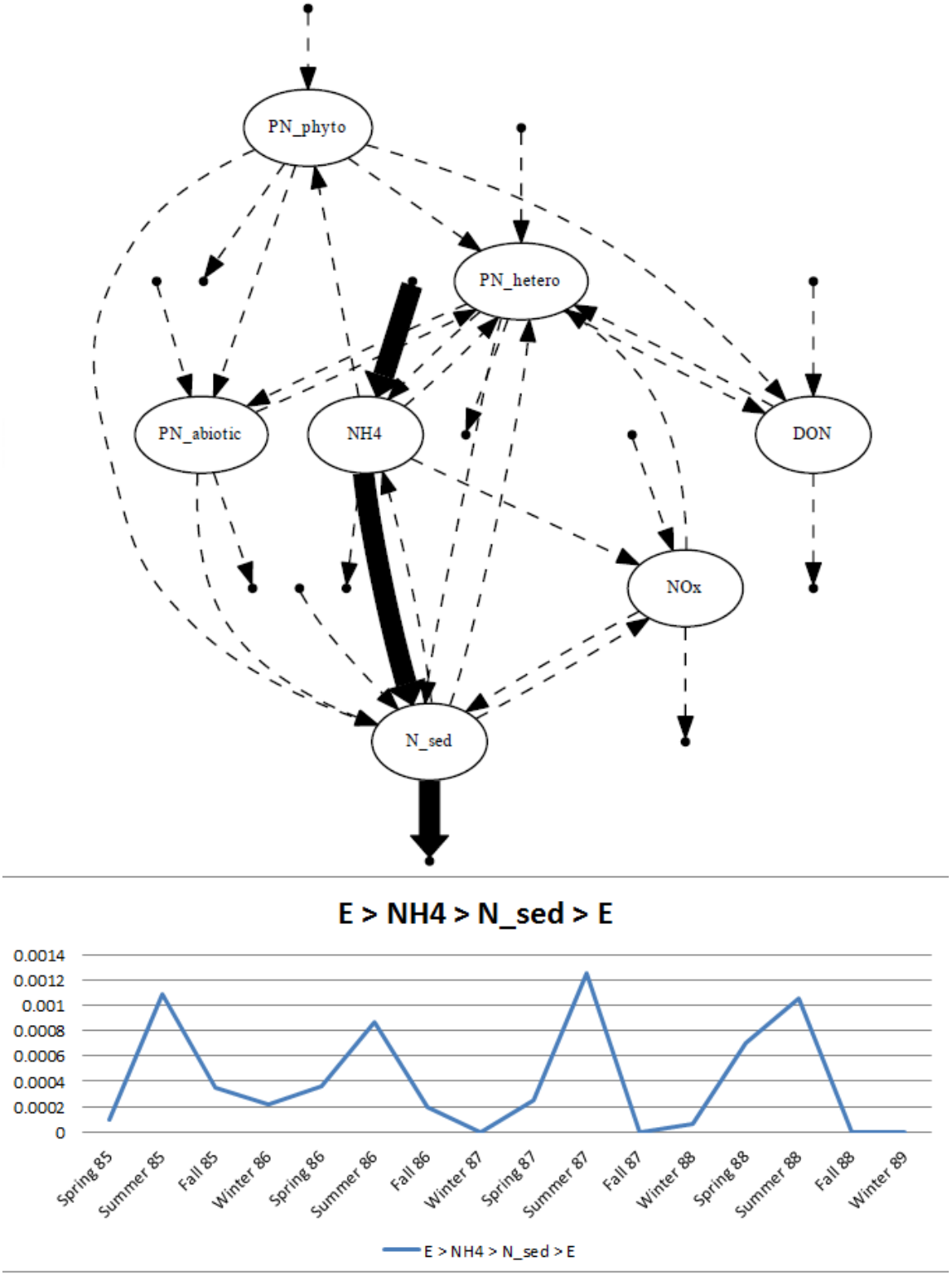


Figure 30: One of the eight fluxes found during feature selection. The line graph shows the flux activation values as they progress chronologically over the time period covered in the data.

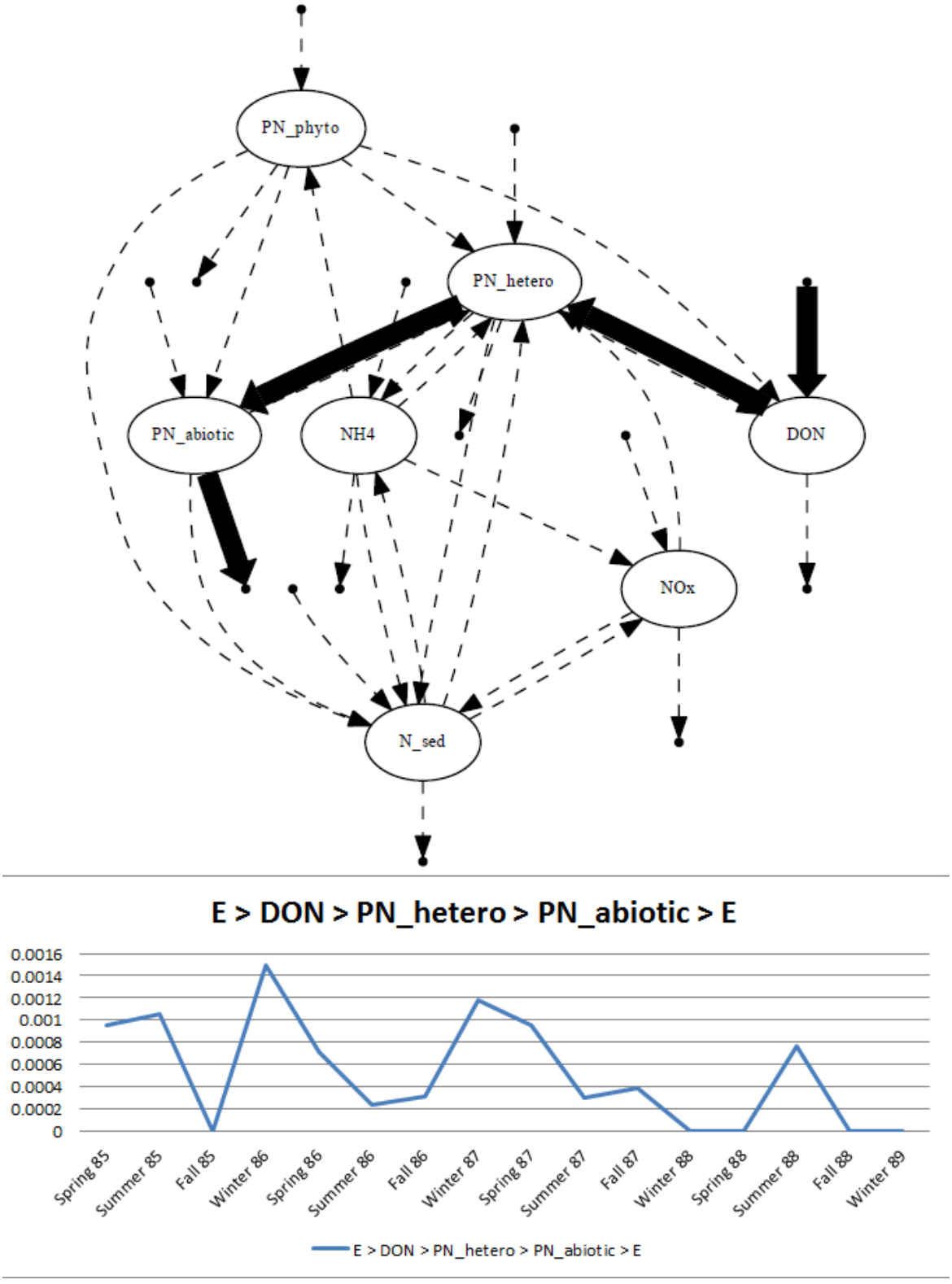


Figure 31: One of the eight fluxes found during feature selection. The line graph shows the flux activation values as they progress chronologically over the time period covered in the data.

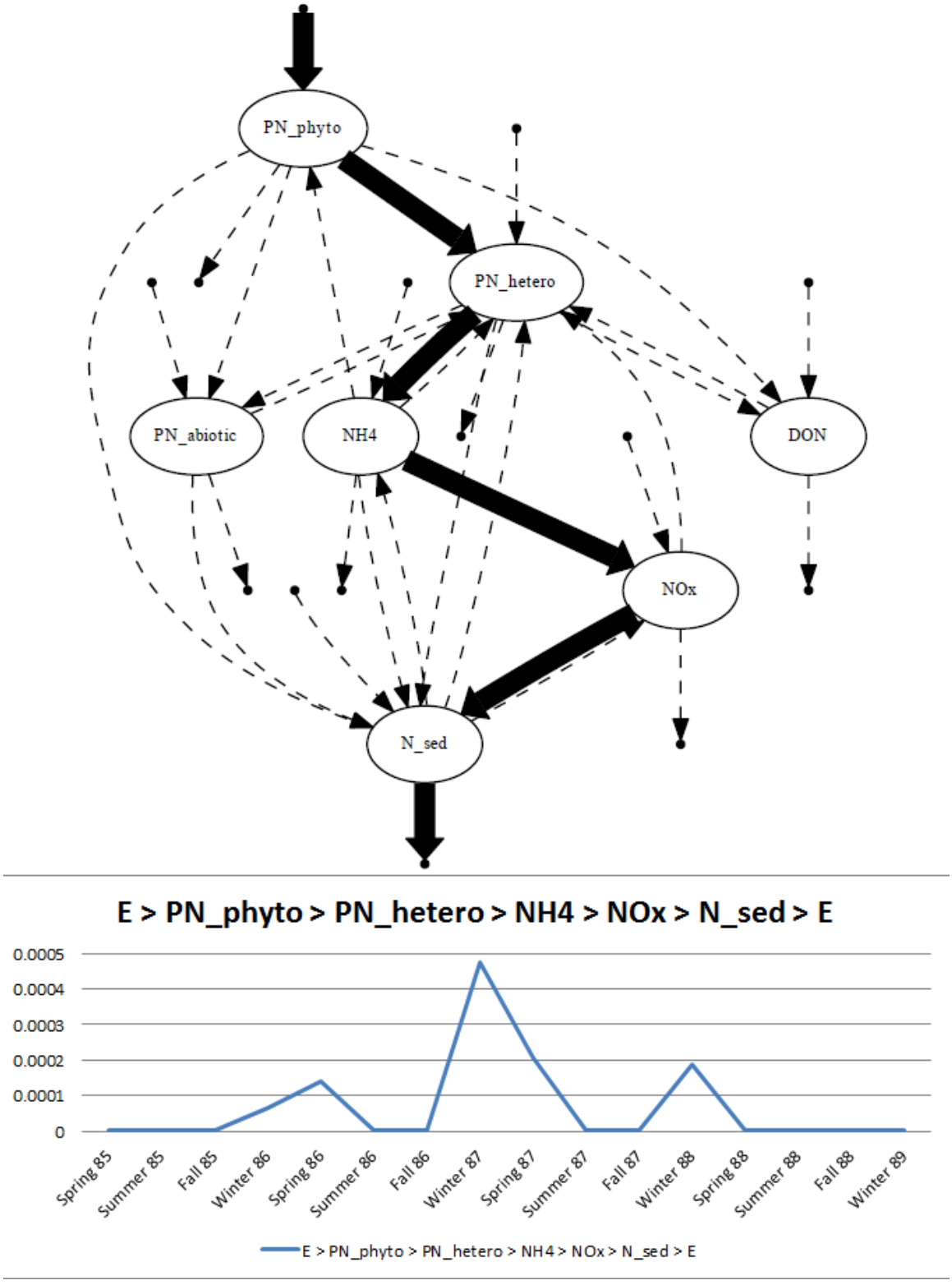


Figure 32: One of the eight fluxes found during feature selection. The line graph shows the flux activation values as they progress chronologically over the time period covered in the data.

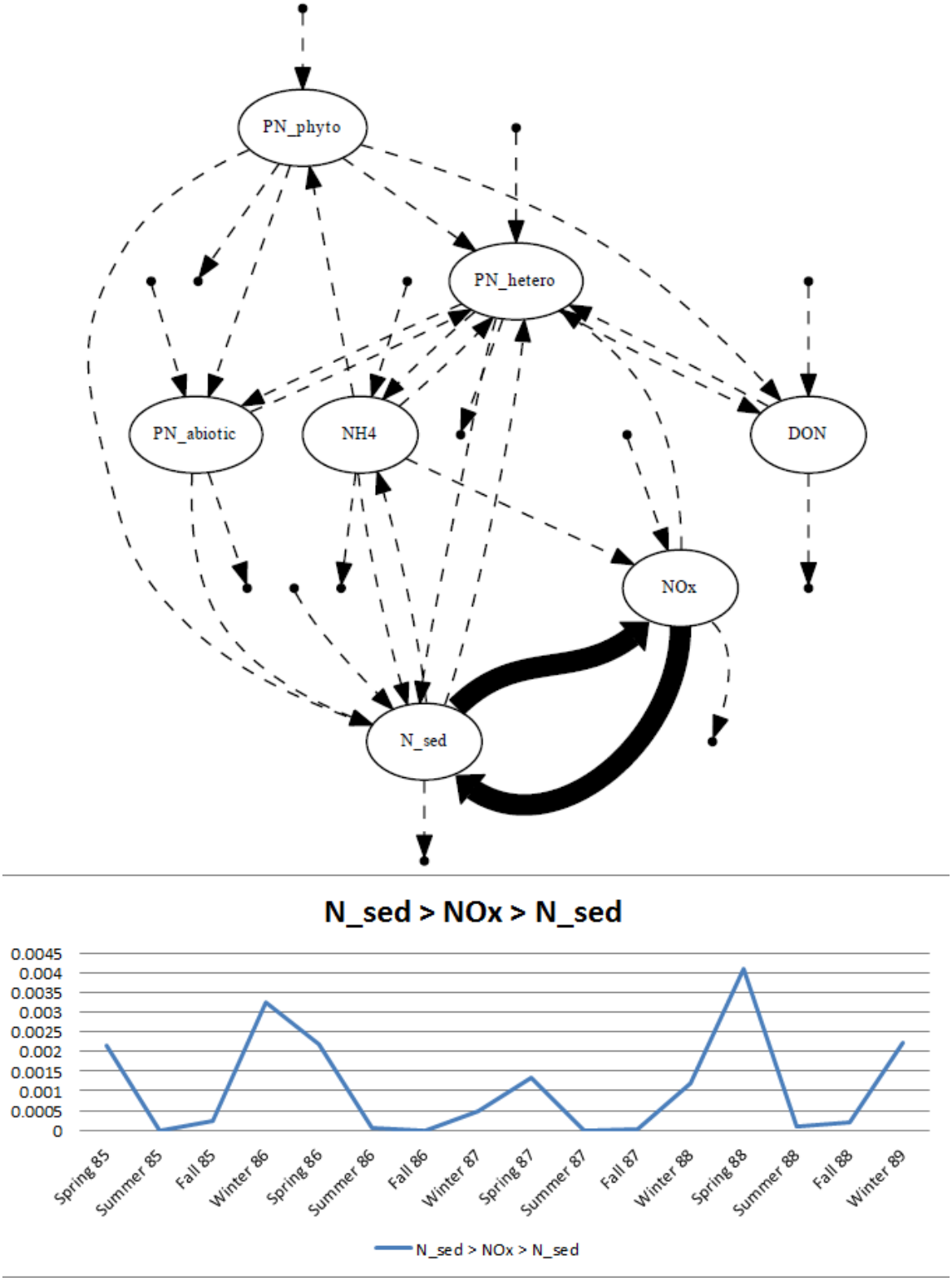


Figure 33: One of the eight fluxes found during feature selection. The line graph shows the flux activation values as they progress chronologically over the time period covered in the data.

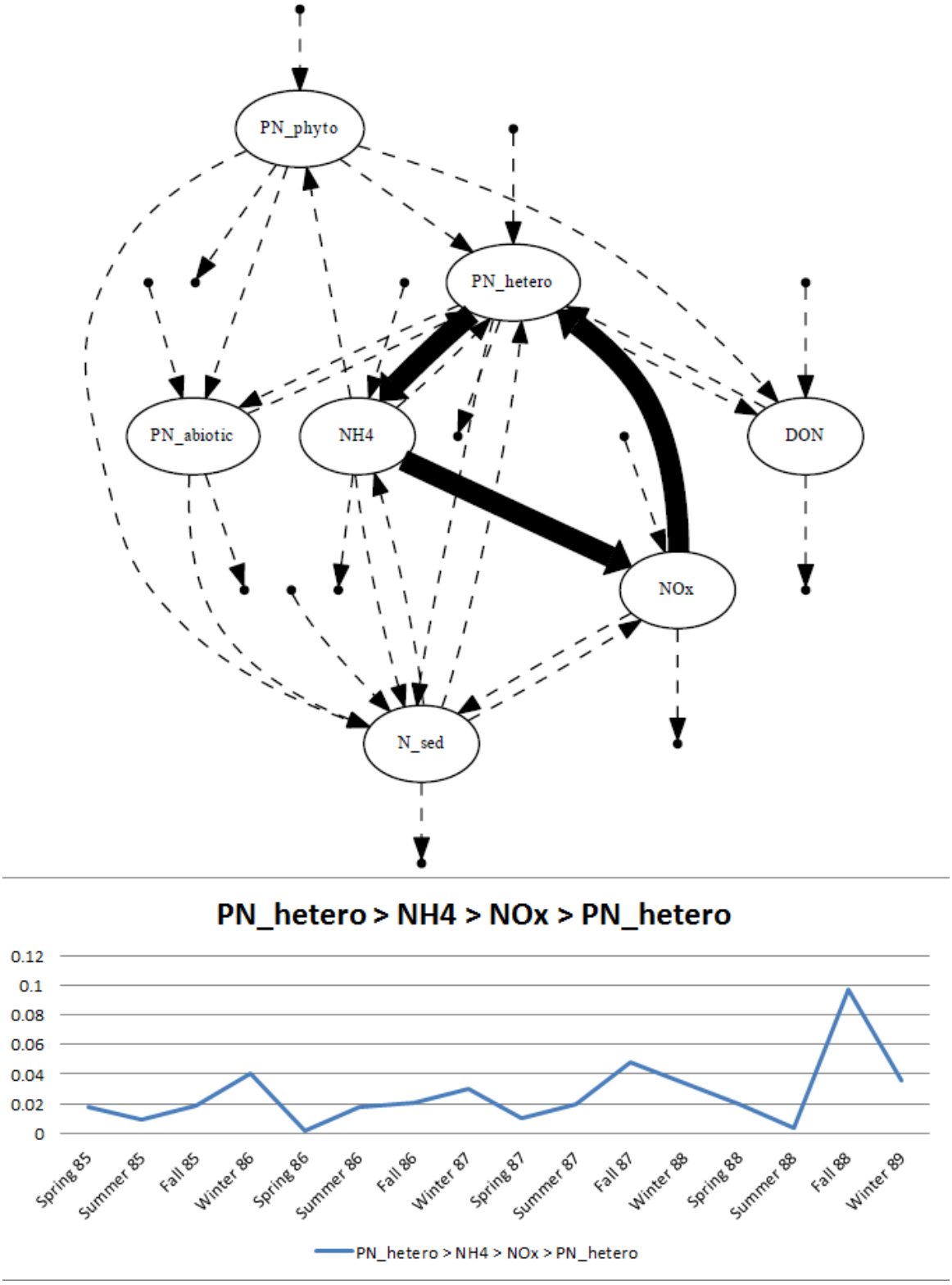


Figure 34: One of the eight fluxes found during feature selection. The line graph shows the flux activation values as they progress chronologically over the time period covered in the data.

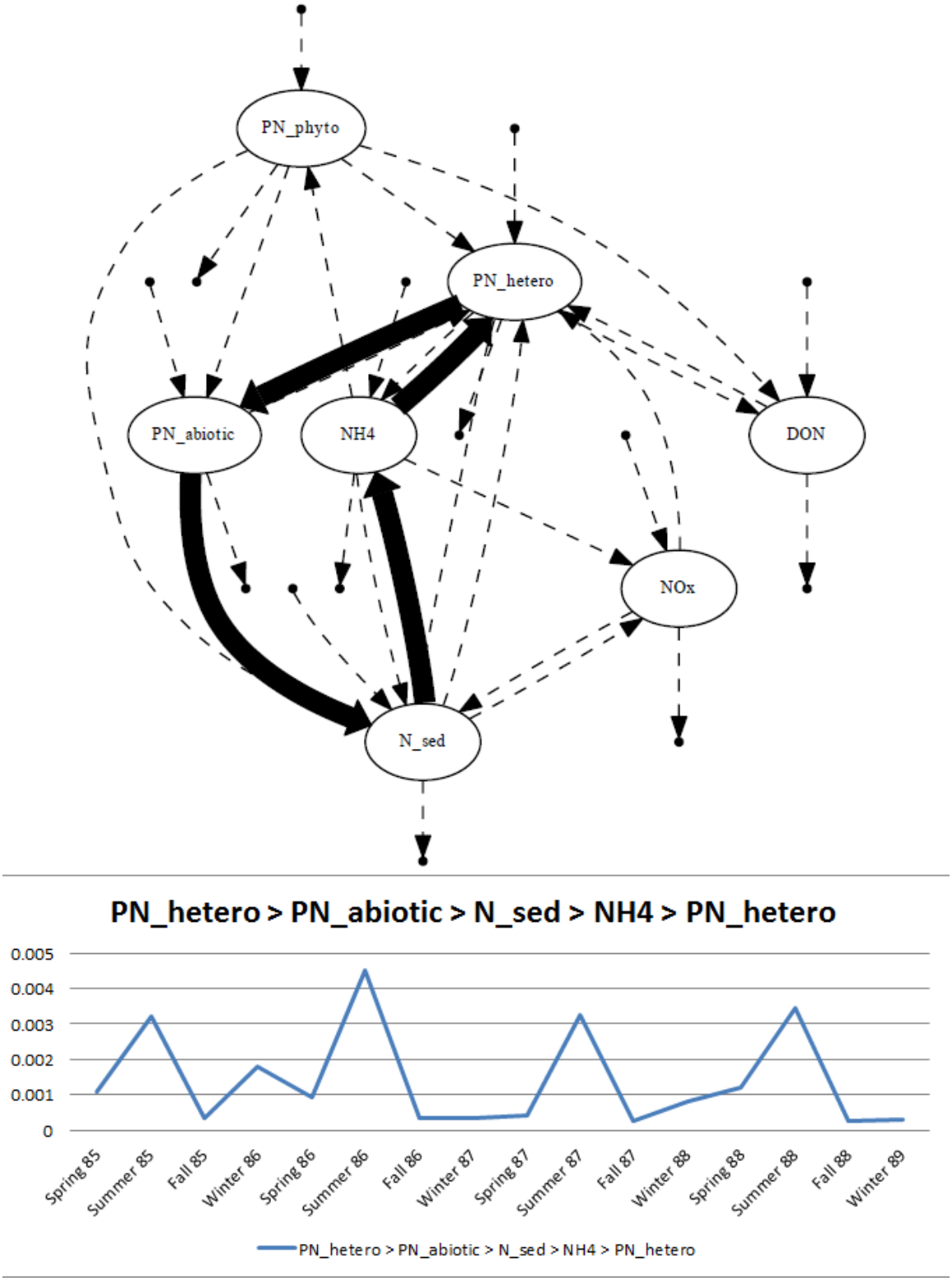


Figure 35: One of the eight fluxes found during feature selection. The line graph shows the flux activation values as they progress chronologically over the time period covered in the data.

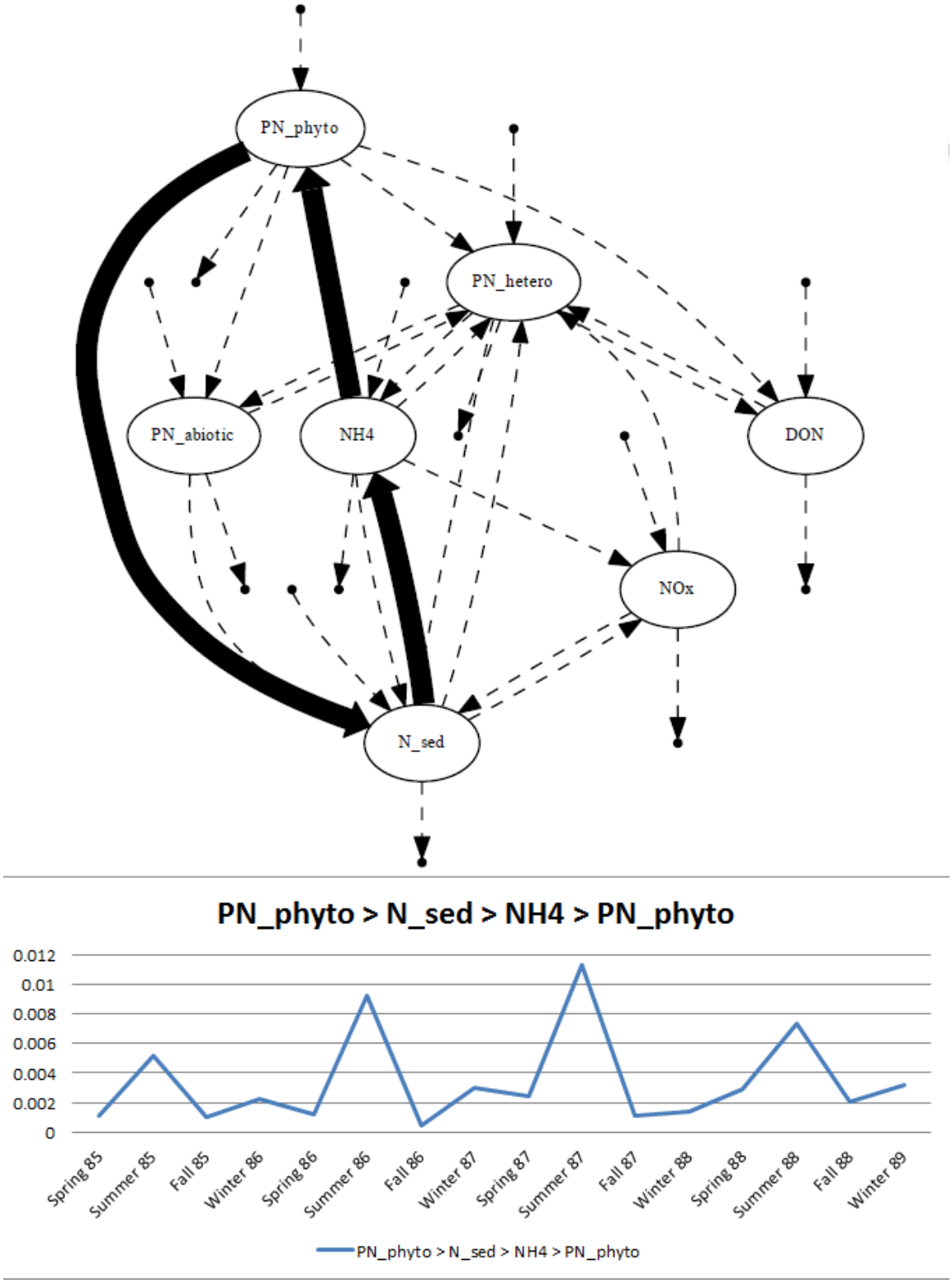


Figure 36: One of the eight fluxes found during feature selection. The line graph shows the flux activation values as they progress chronologically over the time period covered in the data.

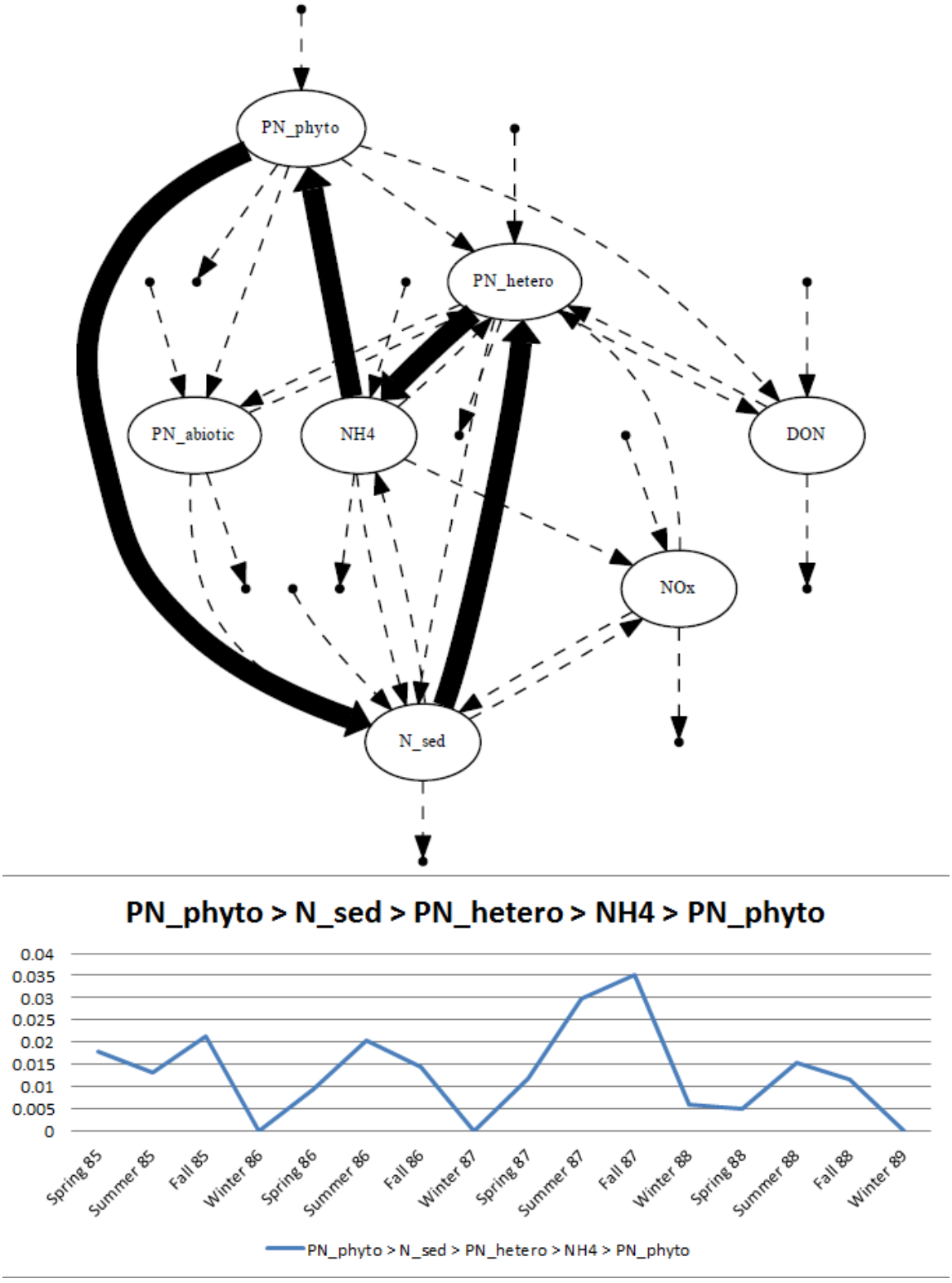


Figure 37: One of the eight fluxes found during feature selection. The line graph shows the flux activation values as they progress chronologically over the time period covered in the data.

In machine learning there are parameters and methodology choices that are inherent to the process of learning from training data. With SVM's these choices include things such as normalization verses standardization of the input data, which type of kernel computation to use, kernel specific parameters (exponent for polynomial kernels and gamma for RBF kernels), and an SVM specific parameter called the complexity parameter. A brief explanation of these follows:

- Normalization of Data

  Applying a Gaussian function to each feature column in the data, mapping that column of data into the range -1 to 1.

- Standardization of Data

  Mapping feature columns into the range 0 to 1 by the following formula:

  $x' = x - colmin / colmax - x$

- Kernel Selection

  It is frequently the case when analyzing feature vectors for classification they will not be linearly separable. The SVM algorithm can map these feature vectors into a feature space of higher dimensionality to analyze for separability in that feature space. This is a common practice in machine learning and is done through the use of a kernel. Kernels can be of different types. For this analysis two kernels were evaluated, the polynomial kernel and the RBF kernel.

  Polynomial Kernel : $K(x, y) = \langle x, y \rangle^p$

  RBF Kernel : $K(x, y) = e^{-(gamma * x)}$ where $x = \langle x-y, x-y \rangle^2$

- Complexity Parameter in SVM algorithms

  The complexity parameter relates to the flexibility given to the SVM to choose hyperplanes. As the complexity parameter increases, the SVM is given greater flexibility to choose marginal hyperplanes that are less maximal in order to minimize training error.

The SVM used to classify the flux vectors for this analysis used a polynomial kernel with an exponent of 1.0, a complexity parameter of 10.0, and normalization was performed on the input vectors. Different parameters were explored such as RBF kernels and polynomial kernels with different exponents, but the parameters chosen performed best for the feature space defined by this analysis.

The results show the proposed methodology can be used for knowledge extraction towards seasonal variation in complex systems. The set of eight fluxes shown above represent a sub-network within the system that can classify which season a model is in with 100% accuracy over the available data.

Table 1 illustrates the accuracy and capabilities of the methodology. In achieving classification results of 100% accuracy, it can be said that the eight fluxes used vary predictably from season to season. These eight features accurately divide the numeric space pertaining to seasonal variation in this model. This information can provide valuable domain-specific information to ecologists for further analysis. To date, the current methodology is the only way to attain this information.

Table 1: Legend below.

# of Instances: the number of NTP paths randomly sampled to build the features used in this particular SVM classification run.
Percent Correct: the number of feature vectors classified correctly as belonging to a particular season.
Recall: proportion of examples which were classified as class *A*, among all examples which truly have class *A*, *i.e.* how much of the class was captured.
Precision: proportion of the examples which truly have class *A* among all those which were classified as class *A*.
FP: Proportion of examples which were classified as class *A*, but belong to a different class.

# of Instances in Functional Decomposition	Percent Correct	Recall	Precision	FP
25	62.5	0.625	0.713	0.125
50	68.75	0.688	0.583	0.104
75	68.75	0.688	0.729	0.104
125	75	0.75	0.817	0.083
250	87.5	0.875	0.888	0.042
500	87.5	0.875	0.888	0.042
250+500	100	1.0	1.0	0.0

An important point shown by Table 1 is that classification results for the decompositions of differing sampling sizes increase in accuracy as the sampling sizes increase. This was expected as the functional decomposition is an approximation of the actual system activity based on the sampling size used. Increasing the sample size elevates the accuracy of the functional decomposition. A more accurate decomposition should yield better classification results as the models are more accurately depicted.

Further Analysis of Results

After the feature selection and subsequent classification of the data, the final eight fluxes were subjected to manual testing to see if they could be further reduced to a smaller set that classified the data correctly. The manual testing involved leaving one of the fluxes out of the feature input data to the SVM, for the purpose of evaluating the degradation in classification accuracy resulting from leaving that particular flux out, see Figure 38.

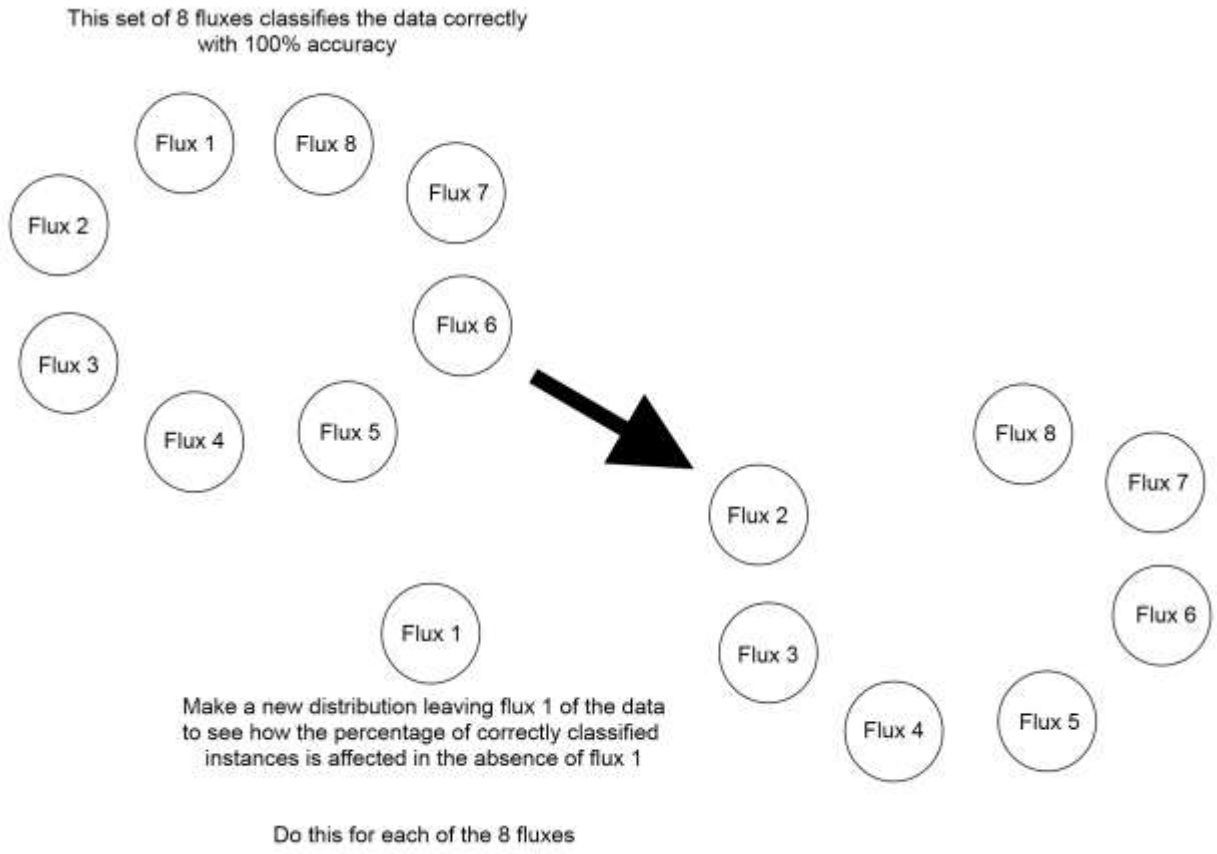


Figure 38: Eight new data sets of seven fluxes are made by removing one of the eight fluxes found during feature selection. These new data sets of seven fluxes will be classified to see how important each of the eight fluxes is towards the classification of seasonality.

There were eight fluxes so eight new distributions of data were created where each of the eight fluxes was removed to form a new distribution. The results of the classifications on the new data sets are below.

Table 2: This table shows the percentage of instances correctly classified when each of the eight fluxes is left out and only the remaining seven are used to classify seasonality for the data set.

Flux being left out	Percentage correctly classified
E > NH4 > N_sed > E	96.88%
E > DON > PN_hetero > PN_abiotic > E	100.00%
N_sed > NOx > N_sed	100.00%
E > PN_phyto > PN_hetero > NH4 > NOx > N_sed > E	100.00%
PN_hetero > PN_abiotic > N_sed > NH4 > PN_hetero	100.00%
PN_phyto > N_sed > NH4 > PN_phyto	96.9%
PN_hetero > NH4 > NOx > PN_hetero	84.38%
PN_phyto > N_sed > PN_hetero > NH4 > PN_phyto	87.50%

The results from manually testing the removal of the individual fluxes in the set of eight shows that it is possible to further reduce to the number of fluxes necessary to retain 100% accuracy in the seasonality classification. There were four fluxes that when left out did not reduce the accuracy of the classification, and upon further investigation three of those four were able to be left out in combination to produce a set of five remaining fluxes that retained the ability to classify the seasonality of the data with 100% accuracy. The five fluxes are listed below.

- E > NH4 > N_sed > E
- N_sed > NOx > N_sed

- PN_phyto > N_sed > NH4 > PN_phyto
- PN_hetero > NH4 > NOx > PN_hetero
- PN_phyto > N_sed > PN_hetero > NH4 > PN_phyto

This set of five fluxes was subjected to another round of manual testing due to the fact that removing features in this manner alters the feature space for the SVM. Further testing was needed to see if these five could be reduced even farther. Manual test was performed again, this time creating five data sets where one of the fluxes was removed from the set of five to create a new distribution with only four fluxes. The results of the manual testing are shown below.

Table 3: This table shows the percentage of instances correctly classified when each of the five fluxes is left out and only the remaining four are used to classify seasonality for the data set.

Flux left being left out	Percentage correctly classified
E > NH4 > N_sed > E	100.00%
N_sed > NOx > N_sed	87.50%
PN_phyto > N_sed > NH4 > PN_phyto	90.60%
PN_hetero > NH4 > NOx > PN_hetero	100.00%
PN_phyto > N_sed > PN_hetero > NH4 > PN_phyto	93.75%

The results from this round of removal testing show the potential to further reduce the number of fluxes needed in seasonality classification. There were two fluxes that when left out did not reduce the percentage correctly classified. Tests were run to see if those fluxes could be left out in combination while still retaining 100% accuracy in classification. The results of the

test showed that both of those fluxes could be left out while still retaining 100% classification on the seasonality of the data sets. The three fluxes are listed below.

- N_sed > NOx > N_sed
- PN_phyto > N_sed > NH4 > PN_phyto
- PN_phyto > N_sed > PN_hetero > NH4 > PN_phyto

The resulting three fluxes were run through another test to see if they could be further reduced, but those tests concluded any further reduction in the number of fluxes broke the ability to retain 100% classification of the data instances. The results of this round of testing are listed below.

Table 4: This table shows the percentage of instances correctly classified when each of the three fluxes is left out and only the remaining two are used to classify seasonality for the data set.

Flux left being left out	Percentage correctly classified
N_sed > NOx > N_sed	81.75%
PN_phyto > N_sed > NH4 > PN_phyto	50.00%
PN_phyto > N_sed > PN_hetero > NH4 > PN_phyto	68.75%

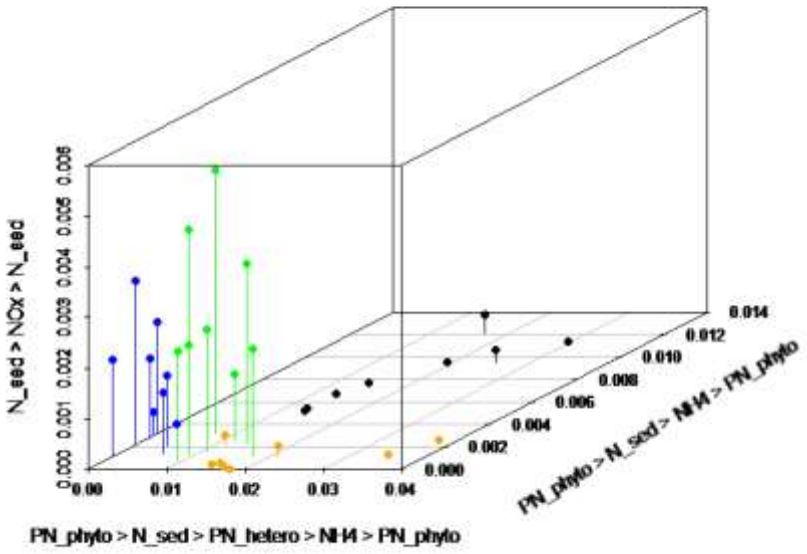
The results of testing the reduction of the set of eight fluxes found from feature selection show that only three fluxes are needed to determine the seasonality of the data instances. These three fluxes can be seen as fluxes involved in encoding the system information regarding seasonality.

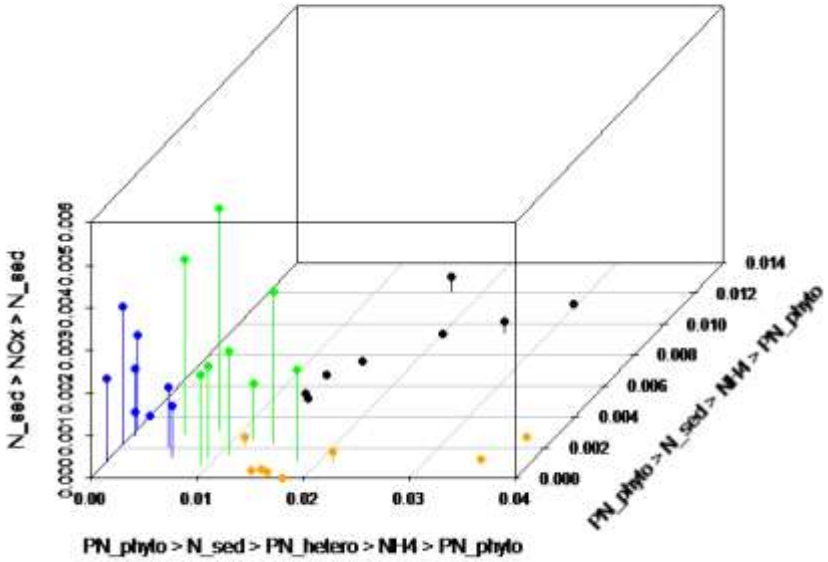
There are three sets of fluxes depicted in the reduction testing, sets of eight, five, and three. While the set of three is the smallest set, the average percentage correct when one of the three is left out is by far the smallest. The average percent correct for the reduction tests show a tolerance level in the event that one of the fluxes has an abnormal value, or is missing. The average percent correct for each of the reduction tests is listed in Table 5.

Table 5: This table shows the average percentage correctly classified for the three rounds of reduction testing. The round testing the set of three fluxes has a significantly lower average when one of fluxes from the set is removed from classification.

Number of fluxes in reduction test	Average percentage correctly classified when one flux is left out
8	95.70%
5	94.37%
3	66.83%

A final analysis of the fluxes that are responsible for encoding the system trait of seasonality would rank the set of three fluxes that retain the ability to classify seasonality with 100% accuracy as the best solution found. However, to some extent the set of five and eight need to be included as secondary sets that show an increased tolerance for dealing with potentially abnormal system activity. Figure 39 shows three-dimensional scatter plots of the feature space created by the set of three fluxes that retain the 100% classification accuracy.


N_sed > NOx > N_sed
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0.00
0.01
0.02
0.03
0.04
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
PN_phyto > N_sed > NH4 > PN_phyto
PN_phyto > N_sed > PN_hetero > NH4 > PN_phyto


N_sed > NOx > N_sed
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0.00
0.01
0.02
0.03
0.04
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
PN_phyto > N_sed > NH4 > PN_phyto
PN_phyto > N_sed > PN_hetero > NH4 > PN_phyto

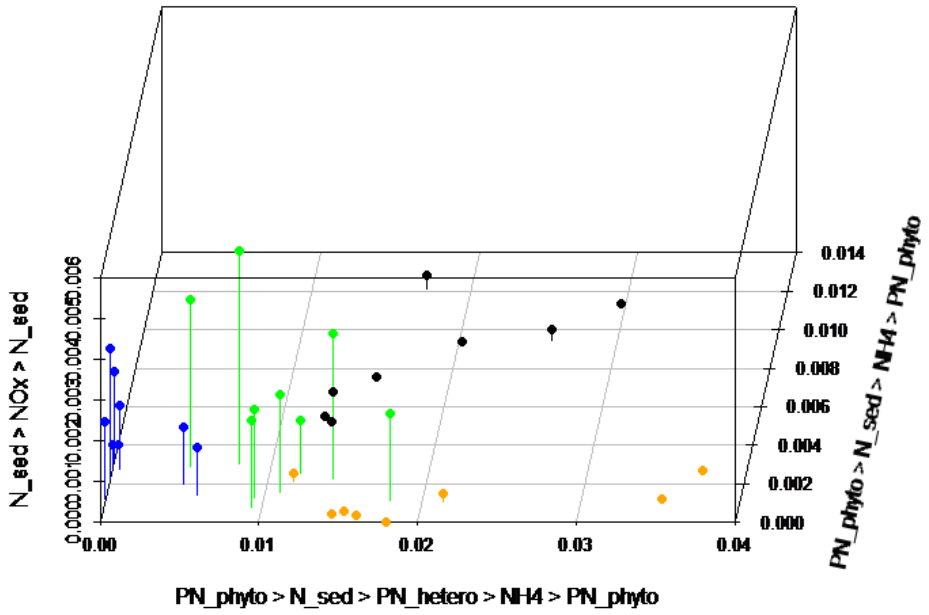


Figure 39: The three figures presented show three-dimensional scatter plots of the feature space created by the set of three fluxes that retain the 100% classification accuracy, black = summer, orange = fall, green = spring and blue = winter. The images show different perspectives of that feature space to illustrate it optimally.

Pattern Discovery in Results

The results from running the functional decomposition methodology on the 16 chronological data sets present a time series for each flux in the system. This allows the activity for each flux to be viewed as a signal that has 16 sampling points, see Table 6 and Figure 40.

The eight fluxes found in the results from running the feature selection methodology for the Neuse River estuary all have a periodic shape when plotted on a line graph as in Figure 40. Intuitively this is reasonable since the data is chronological and the seasons repeat themselves at a concrete interval, but an analysis of fluxes outside the eight found through feature selection

show less conformity to a periodic shape. Figures 41, 42 and 43 show the comparison between the eight fluxes found and eight fluxes randomly chosen from the rest of the decomposition values.

Table 6: This table shows the decomposition values for the flux *E > NH4 > N_sed > E* over the chronological seasons in the available data.

Season/Year	E > NH4 > N_sed > E
Spring 85	0.000093
Summer 85	0.001092
Fall 85	0.000345
Winter 86	0.000215
Spring 86	0.000356
Summer 86	0.000863
Fall 86	0.000198
Winter 87	0.000001
Spring 87	0.000249
Summer 87	0.001253
Fall 87	0.000001
Winter 88	0.00006
Spring 88	0.000707
Summer 88	0.001057
Fall 88	0.000001
Winter 89	0.000001

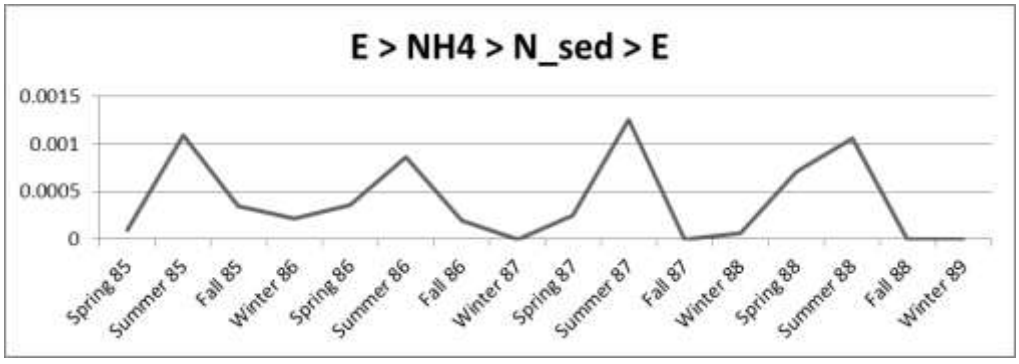


Figure 40: This figure shows the decomposition values for the flux *E > NH4 > N_sed > E* plotted as a time series on a line graph.

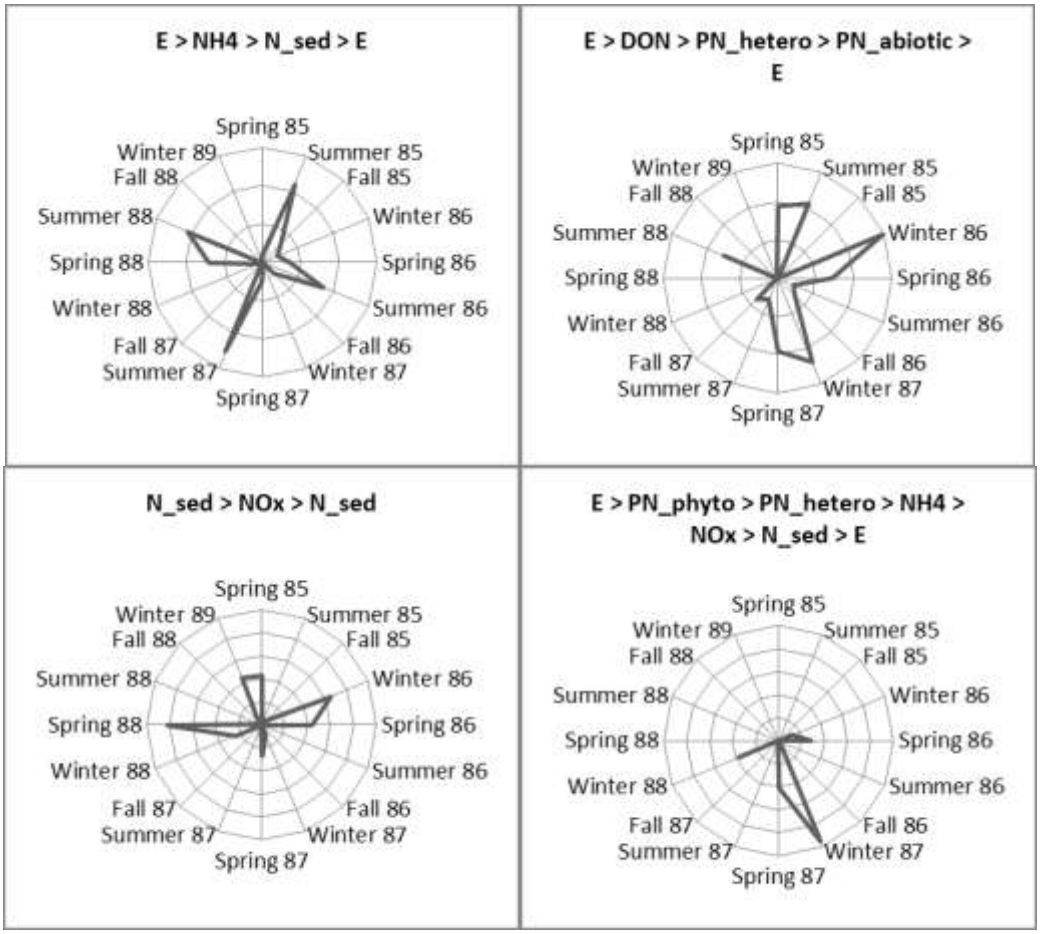


Figure 41: This figure shows the first four out of the eight fluxes found during the feature selection methodology on the Neuse River estuary in North Carolina. Each of the fluxes is graphed on a radar plot where each section represents a specific year and season. Since there are four years of data and four seasons in each year the radar plot is optimally suited to graph this particular data set. The preiodic nature of the flux values are easily noticable in this graphing format. The oscilation of the lines in these graphs seems to flucuate seasonally, peeking at similar seasons each year which could help the SVM distinguish seasonality with greater accuracy.

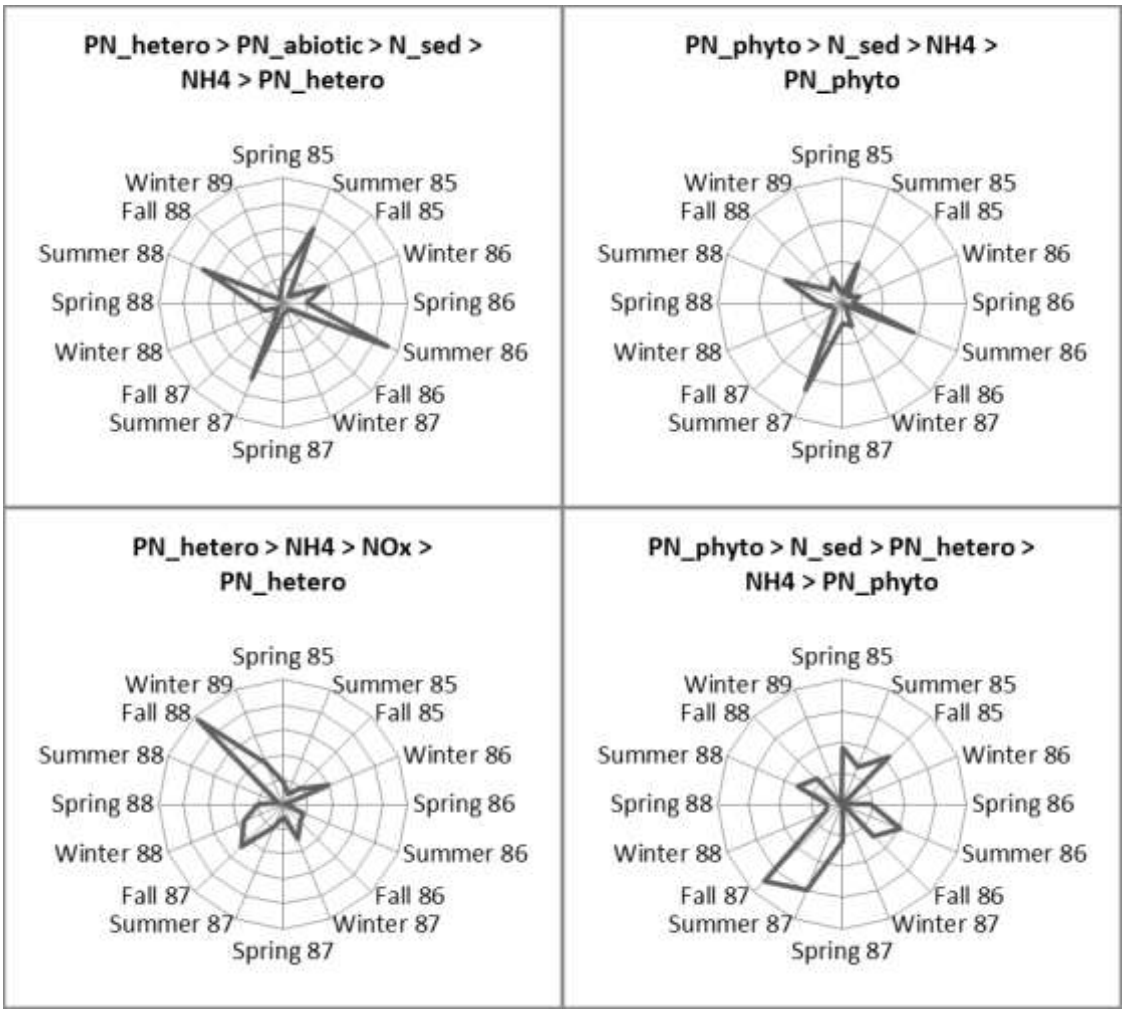


Figure 42: This figure shows the second four out of the eight fluxes found during the feature selection methodology on the Neuse River estuary in North Carolina. Each of the fluxes is graphed on a radar plot where each section represents a specific year and season. Since there are four years of data and four seasons in each year the radar plot is optimally suited to graph this particular data set. The preiodic nature of the flux values are easily noticable in this graphing format. The oscilation of the lines in these graphs seems to flucuate seasonally, peeking at similar seasons each year which could help the SVM distinguish seasonality with greater accuracy.

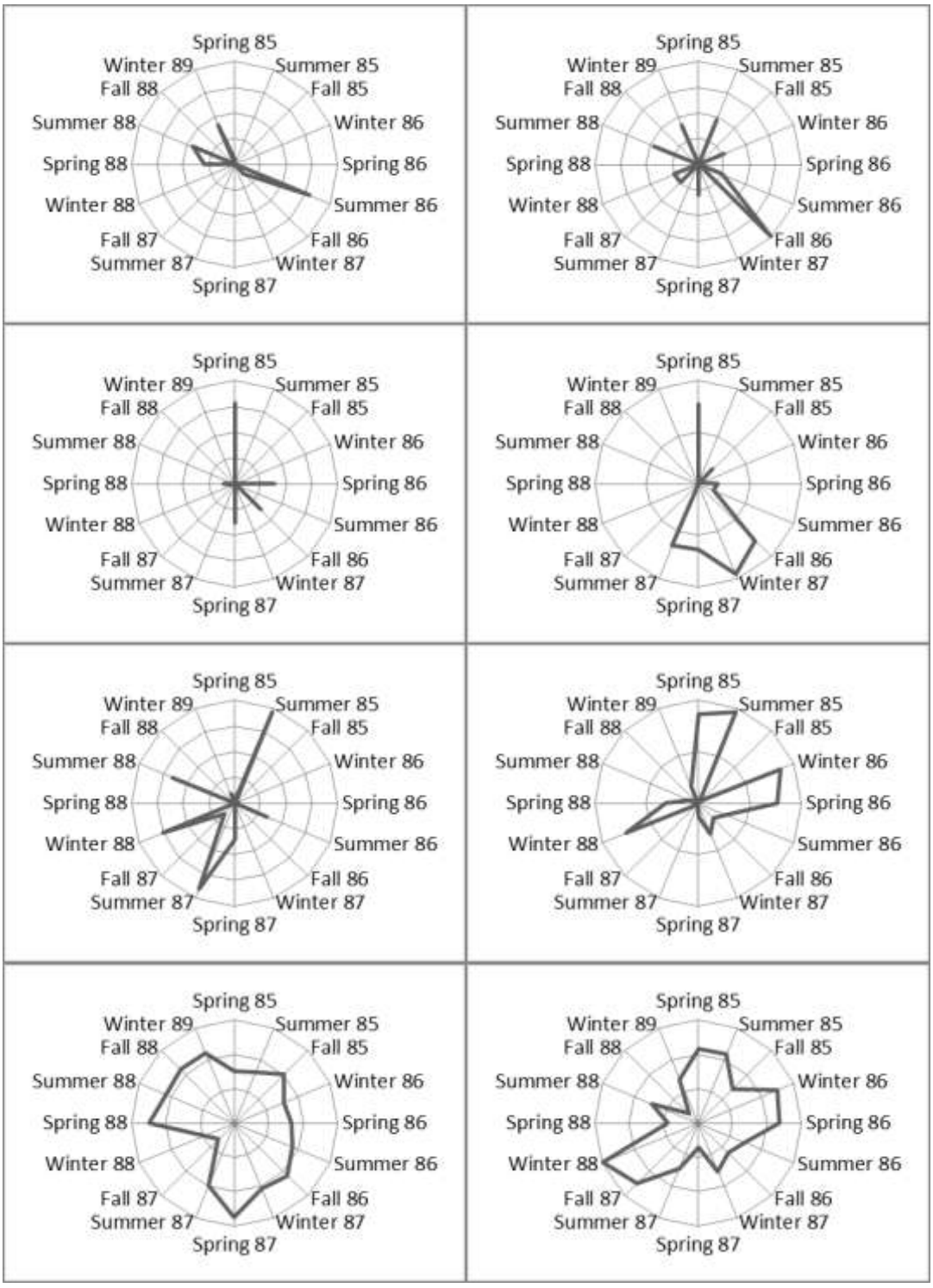


Figure 43: This figure shows randomly selected fluxes from the data distribution outside of the eight found during feature selection. The patterns presented by these fluxes are much less periodic in nature. There is no clear cut pattern that emerges from these radar plots as opposed to the periodic pattern that emerges from the eight fluxes found during feature selection.

A visual comparison between the eight fluxes found during feature selection and the other eight fluxes in the figures above clearly shows a more periodic nature in the signals from the eight fluxes isolated in feature selection. This was not expected apriori, but it is reasonable that feature selection would isolate signals presenting a more clearly defined pattern expressing itself with the same frequency as the concept being learned. Treating the flux time series as signals opens a new venue for data mining on a wide variety of tasks. Signal processing is a well-defined field with robust methodologies suited for a multitude of different data mining tasks. The decomposition routine outlined herein, applied over chronological data sets, transforms the adjacency matrices encoding the models being analyzed into signals expressing various activations and frequencies. These signals serve well for knowledge extraction and data mining as exemplified herein towards seasonality classification.

Ecological Analysis

The results produced by the proposed methodology allow ecological analysis not previously possible. One point that is particularly interesting is that all of the diagnostic fluxes of the Neuse River Estuary (NRE) models contain the N-Sediment compartment. Both Network Environ Analysis - Storage (NEA-S) and empirical studies provide evidence that N-Sediment is the key "capacitor" (storage and release site) for nitrogen in this system.

The x3-Sed compartment behaves like a harmonic oscillator in sequestering nitrogen during winters and, capacitor-like, releasing it to biogeochemical processes during summers [116].

Three observations made from the three-dimensional plot (Figure 44):

1. The seasonal partitioning is clear.
2. The sequester-release capacitance is evident in the winter > spring versus summer > fall data points (high values on the N-sed > NOx> N-Sed flux axis for winter > spring versus low values for summer > fall).
3. The seasonal sequence (spring > summer > fall > winter) can be visually traced in the space, and summer appears to be the most distinctive.

The summer peaks in scores from the flux analysis, particularly for the two fluxes containing N-Phyto, are consistent with this interpretation of the seasonal mechanisms of summer nitrogen uptake and winter nitrogen sequestration in the NRE. The peaks for the N-sed > NOx > N-Sed flux are primarily in winter and spring.

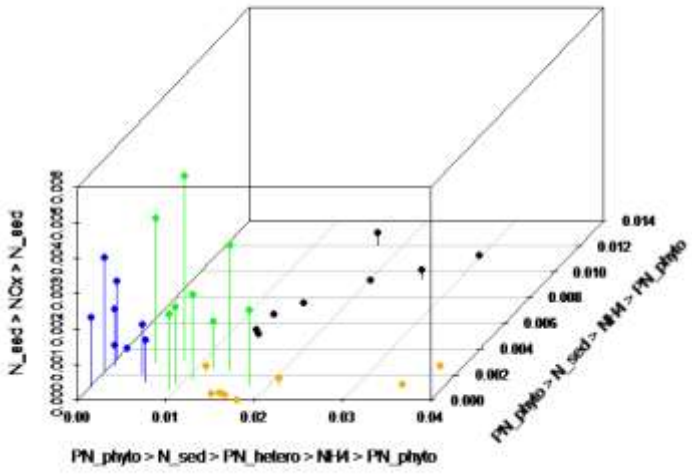


Figure 44: This figure shows a three-dimensional scatter plot of the feature space created by the set of three fluxes that retain the 100% classification accuracy as presented in the results, black = summer, orange = fall, green = spring and blue = winter.

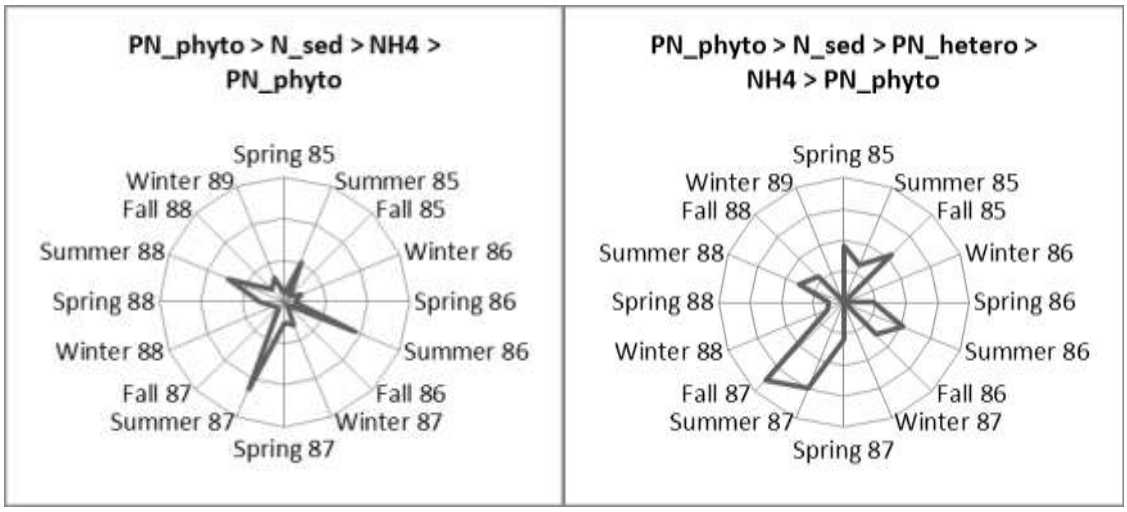


Figure 45: This figure shows two fluxes graphed on a radar plot where each section represents a specific year and season. Since there are four years of data and four seasons respectively in each year, the radar plot is optimally suited to graph this particular data set. The periodic nature of the flux values are shown optimally with this graph style. The oscillation of the lines in these graphs seems to fluctuate seasonally, peaking at similar seasons each year which could help the SVM distinguish seasons with greater accuracy.

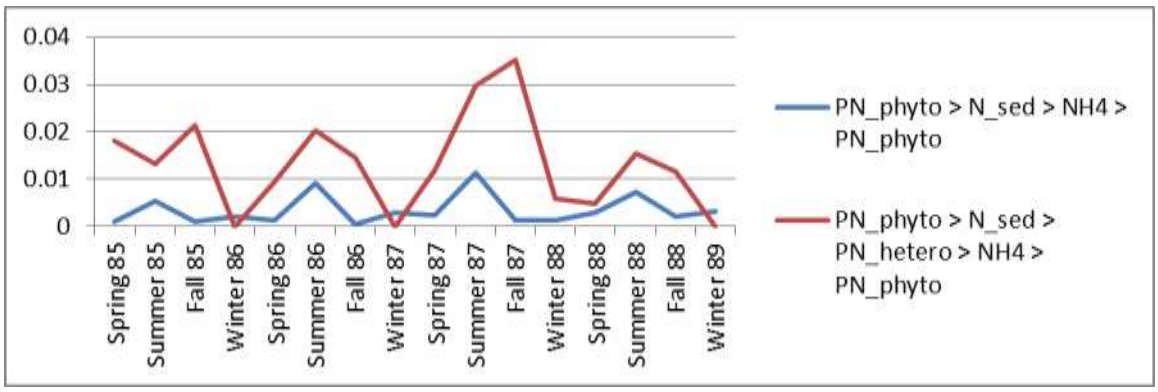


Figure 46: This figure shows two fluxes graphed on a line plot chronologically over a four year period.

In winter throughflows are low, signifying small nitrogen exchange with other compartments; in summer throughflows are high, indicating active interchange between sediment and the other compartments. Note that unit-input driven sediment environs reflect this pattern, storing nitrogen during winter and releasing it to active biogenic processes during summer. However, measured inputs change this picture in absolute environs, to one dominated by the dissolved nutrient compartments, x5-NOx, x4-DON, and x6-NH4. The three diagnostic fluxes found for the NRE models contain three of the four dominant storage compartments from the NEA-S results, N-Sed, NOx, and NH4.

Network analysis by Christian and Thomas [116, 117] showed that, on average, half the nitrogen needs of phytoplankton are met by nitrogen that once resided in sediments. Phytoplankton dependence on sediment nitrogen varied seasonally, ranging from 5–32% in winter to 71–85% in summer [117]. This reflects regeneration and mass transport mechanisms that transform nitrogen and move it from the sediment pool of winter up into the water column where it becomes seasonally available to support phytoplankton growth. Empirical evidence showed that nitrogen from sediments is released during the summer months as ammonium and taken up by phytoplankton [120, 121]. This corresponds with the information presented in Figure 45 and Figure 46.

Corbett [118] empirically demonstrated the key role of sediment in the nitrogen dynamics by demonstrating the diffusive and advective (resuspension) of nitrogen from sediments producing an internal loading to the NRE as great, if not greater, than the external watershed loading. Corbett [118] reported that even though watershed nitrogen loading of the NRE had been reduced by 30%, total nitrogen in the NRE remained the same. The internal nutrient loading from sediment was proposed as the major reason for total nitrogen showing no reduction

during nutrient loading reduction. The presence of the N-Sed compartment in all of the diagnostic fluxes of the NRE models indicates its key role in the N-dynamics that these models represent. All the diagnostic fluxes of the Neuse models presented in the results contain the N-Sed compartment. Both storage NEA and empirical studies provide evidence that N-Sed is the key capacitor (storage and release site) for nitrogen in this system.

Shape and variability of the N-Sed > NOX > N-Sed line graph (lower and less distinctive peaks, less variability) is much different from the shape and variability of the two N-Phyto line graphs (higher and more distinctive peaks, more variability).

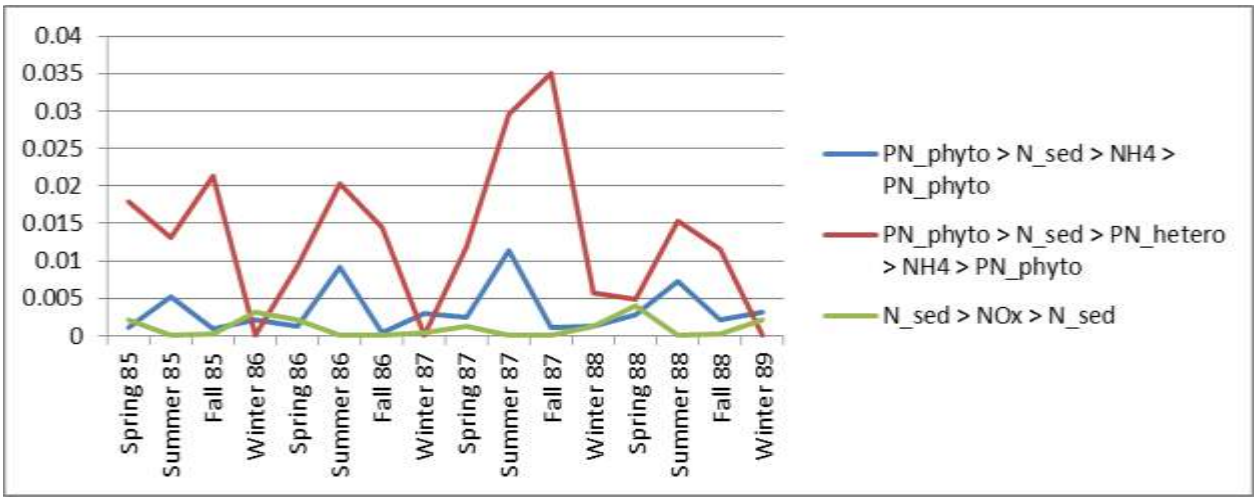


Figure 47: This figure shows three fluxes graphed on a line plot chronologically over a four year period.

This dichotomy is represented in the NEA results for the NRE in the graphs of seasonal total environ throughflow (TET), total system throughflow (TST), total environ storage (TES) and total system storage (TSS) sequences over the 16 season time series. One clear difference in

the throughflow vs. storage data is that there is a great deal more variability in the TET and TST sequences than the TES and TSS sequences demonstrated by the much higher coefficient of variation values for TET and TST sequences; this reflects inertial damping associated with accumulated stocks [116]. Also, in Figure 46, the two fluxes starting with N-phyto are biotic (N-phyto sequences), and the flux starting with N-Sed is abiotic.

Another point that can be drawn from ecological analysis is the presence of NH4 and the absence of NOx in two of the N-Phyto cyclic fluxes. Corbett [118] presented arguments for why NH4 had shown an increase in concentration during the period of N-loading reduction to the NRE. In estuaries such as the NRE, which are organic-rich, eutrophic, and show episodes of anoxia, NH4 is recycled directly back to the water column from the sediments with the occurrence of denitrification [118, 119]. The increase in recycling of NH4 may also be related to greater mineralization of organic matter [118]. Both of these pieces of evidence are congruent with the compartment make-up of two of the N-Phyto fluxes found for the NRE nitrogen models.

In addition to the points mentioned above, the methodology and presented results brings up interesting points. A few of these are listed below.

1. The three diagnostic fluxes isolated in the results to classify seasonality of a system with 100% accuracy have five of the seven compartments. Why are N-Det (particulate organic nitrogen) and DON (dissolved organic nitrogen) missing?
2. In the diagnostic fluxes, it's not the compartments, per se; it's their sequences that appear to have the diagnostic power. All of the 640 fluxes are made up of the same compartments, but have different sequences. This connects strongly to the

idea from environs that sequence is the essential diagnostic feature of network function.

3. It is our current understanding that systems have many traits. In this case season was an example trait. Each trait will be diagnosed by a distinctive flux set. Running the presented methodology over different traits should present opportunities for meaningful data analysis based on which traits are selected. In order to run the proposed methodology on a different trait, example models reflecting the possible values of the trait need to be provided. The methodology's ability to extract diagnostic fluxes comes from its ability to compare the same structural model with different flow values representing potential trait values. In the Neuse River Estuary the models given to the methodology were representative of the four different seasonal states for the system (there were four models each for winter, summer, fall and spring, totaling 16 models). This means diagnostic fluxes can be discovered for a trait as long as the differing flow values for the models can be ascertained.

The methodology and results presented in this work pose a new approach to system analysis that has revealed exciting ecological perspective. The ability to construct an encoding framework for complex systems similar to what genetics does for biological systems transformative possibilities for computational analysis. This ability to see how individual steady state components of a system are involved in encoding and responding to higher level system traits bridges a gap between system complexity and system structure in ways not previously possible.

Classification Using the Adjacency Matrix

As an alternative to using the functional decompositions as feature vectors into the SVM, the weighted adjacency matrix representation of the system can also be used. In this approach each arc in the flow-weighted adjacency matrix is used as a feature presented to the SVM. The weighted adjacency matrices for this approach were built from the 250 and 500 sampling size distributions of data simulated from NPT. Each time an arc appeared in a path, it was logged in the adjacency matrix. The classification results for this process are shown in Table 7. Feature

Table 7: Legend below.

# of Instances: the number of NTP paths randomly sampled to build the features used in this particular SVM classification run
Percent Correct: the number of feature vectors classified correctly as belonging to a particular season
Recall: proportion of examples which were classified as class *A*, among all examples which truly have class *A*, *i.e.* how much of the class was captured.
Precision: proportion of the examples which truly have class *A* among all those which were classified as class *A*.
FP: Proportion of examples which were classified as class *A*, but belong to a different class.

# of Instances for Weighted Adjacency Matrix	Percent Correct	Recall	Precision	FP
250	93.75	0.938	0.95	0.021
500	100	1.0	1.0	0.0

selection was attempted on these data sets, and the best subset from the 64 possible features from the eight by eight adjacency matrices were the 36 non-zero arcs in the matrices. Some arcs have a low probability of occurrence and did not occur in the sampled pathways; these arcs were not in the 36 selected features. Any further reduction in features beyond these 36 degraded the

results. Multiple SVM settings were applied and the best settings found were a polynomial kernel with an exponent of 1.0, a complexity parameter of 10.0, and normalization of the input vectors. These settings were the same settings used for classifying the flux coefficients presenting good opportunity for comparative analysis.

The results for this approach achieved 100% accuracy. This is the same percentage as the approach using the functional decomposition vectors, however this approach performed better on the corresponding sampling sizes in comparison with the other approach. The other approach needed to combine the 250 and 500 sampling size data sets in order to achieve 100% accuracy. The fact that this approach attains 100% accuracy with the 500 sampling size data set and the functional approach only attains 87% accuracy has potential to be misleading. The fact that 100% classification results can be attained at all means this is a relatively easy concept to classify. The features pertaining to the respective seasons fluctuate in such a way that the SVM has an easy time classifying data instances correctly. This approach outperforming functional decomposition on an easy problem does not mean it will outperform functional decomposition on a harder problem. An example of a harder problem could be trying to classify the year of a data instance rather than the season. Ecologically, this classification problem might be irrelevant, however it could serve to illustrate the methodologies potential towards being applied to challenging problems. The algorithmic runtime for this approach is linear in number of edges appearing in the simulated data, and thus scales much better than functional decomposition. Another difference between the two approaches was the number of features needed. The functional decomposition was able to use feature selection to perform classification based on a smaller set of features, isolating more important features and suppressing ones considered noise.

With the adjacency matrix approach all the non-zero arcs from matrix needed to be used, any reduction of this feature set degraded the accuracy of the classification results.

Reversing the Transformation

Feature selection on the coefficient vectors from the functional decomposition isolates the information in the domain of the functional decomposition that is necessary for optimal classification of the data sets in the distribution. The eight features isolated using feature selection in the present analysis can be used in a reverse transform to map only these coefficients back to the primary domain of the flow-weighted adjacency matrix. The resultant adjacency matrix contains only the portions of flow for an arc embedded in the diagnostic fluxes determined to predictably fluctuate from season to season. The matrix produced by the reverse transform is a legitimate system that is synthetically produced. For an additional comparison between the adjacency matrix domain and the domain of the functional decomposition, the 250+500 data set that achieved 100% classification accuracy had this reverse transform applied to obtain modified adjacency matrices for each of the 32 data sets (four years with eight seasons each) using only the eight coefficients found through feature selection. The resultant adjacency matrices were grouped into a distribution of flow vectors that were classified seasonally using the same methodology as the original adjacency matrices (36 features sent through an SVM). The classification accuracy on these modified adjacency matrices decreased substantially. The results are shown in Table 8. Feature selection was performed on these 32 input vectors to see

Table 8: Legend, see Table 7.

# of Instances for Weighted Adjacency Matrix	Percent Correct	Recall	Precision	FP
250+500	68.75	0.688	0.696	0.104

if the classification results could be improved but 68.75% accuracy was the highest that could be achieved.

The degradation in accuracy on the adjacency matrices after the reverse transform was applied implies a difference between the ways information is presented between the two domains. The only information needed in the domain of the functional decomposition is the eight fluxes found through feature extraction, when this isolated information is mapped back into the original domain (adjacency matrix domain) the presentation of the information is such that a machine learning algorithm cannot mathematically reproduce the results from the domain of functional decomposition. The adjacency matrices produced by this reverse transform are reasonable steady state systems that contain only the portions of network flows isolated computationally as varying predictably between the different seasons (in much the same way a band pass filter works on an audio signal). Further research needs to be done, but there is a distinct difference shown here in the way the two domains present information.

Application Towards Classification of Data Sets by Year

Seasons present a substantial and predicable change to an ecosystem. Due to seasonal fluctuation there are different affecters presented to the ecosystem such as temperature differences, length of daylight, and plant growth. The data given to the knowledge extraction methodology presented herein has affectively been able to extract information for and classify data according to season. Since the ecosystem has predictable and pronounced changes brought on by season, this process of seasonal information extraction is capable of performing with 100% percent accuracy. To see how the methodology performs for classification towards a harder problem, the same data sets were used towards classifying which year a data instance belonged

to. The years consisted of four seasons and started with spring. The years and corresponding seasons are presented in the Table 9.

Table 9: This table shows how the different season/year data instances were grouped into year classifications.

Year One	Spring 85, Summer 85, Fall 85, Winter 86
Year Two	Spring 86, Summer 86, Fall 86, Winter 87
Year Three	Spring 87, Summer 87, Fall 87, Winter 88
Year Four	Spring 88, Summer 88, Fall 88, Winter 89

Classifying which year an instance belonged to turned out to be a harder problem than seasonal classification. Where the seasons present clear classifications regions with drastically different system activities, yearly separation of data instances does not have the same stark differences to aid the SVM in classification. The tests run to judge the yearly classification problem took the same steps used towards the seasonal methodology presented in this work. Feature selection was performed to isolate an optimal set of features to use in the classification problem, and then leave one out cross fold validation was performed to verify the percent of instances correctly classified. The tests were run using the 500 instance sampling size data sets for both the adjacency matrix and functional decomposition data. Using the adjacency matrix the highest percentage correctly classified was 25%. In comparison, the functional decomposition methodology was able to achieve 62.5% accuracy in classification. The results are presented in Table 10.

The results of running this test show that for certain problems, it is possible for NFA to significantly outperform using the adjacency matrix as feature input in machine learning.

Table 10: This table lists the results of the two classification tests. The data set of instances built from the functional decomposition data performed with higher accuracy.

Method	Percent Correct	Recall	Precision	FP
Weighted Adjacency	25.00%	0.25	0.244	0.25
Functional Decomposition	62.50%	0.625	0.692	0.125

Conclusion

This work has shown how functional decomposition of complex systems can be used for data mining and machine learning. In the analysis carried out on the 16 seasonal data sets from the Neuse River ecosystem, the domain of functional decomposition was used to isolate three fluxes from the ecosystem that can be used to mathematically infer what season the ecosystem's data were drawn from. Using the domain of functional decomposition and these three fluxes, a Support Vector Machine was used to classify the data sets as belonging to a particular season. This technique was compared with another method where the arcs flows in the adjacency matrices of these systems were used as features given to a Support Vector Machine. In both these methods 100% accuracy of classification was achieved.

The ability to achieve 100% accuracy with only three fluxes from the Neuse River estuary system illustrates how basic relationships within a complex system can encode high level system traits. A disconnect is typically present between compositional elements of a system and the organized behaviors and activities those compositional units engage in during interaction with each other. The methodology outlined herein presents fluxes as a missing layer of compositional organization within complex systems that provides the ability to bridge this disconnect. Using fluxes together with machine learning and data mining, a methodology called

Network Flux Analysis (NFA) was constructed. The key steps in NFA are defining a high a level system trait or function (*i.e.* seasonality), constructing data sets illustrating the system at the various state values expressed by this trait, and using feature selection and machine learning to extract which compositional units within the system are encoding the trait values. This framework of information encoding is reminiscent of how genetics works to assess encoding within biological systems. In biological system there are base compositional units called nucleotides that group together into secondary compositional units called codons and these codons group together into a third compositional unit called genes that encode high level biological traits and functionality such as eye or hair color. In similar fashion to this, the NFA has three levels of compositional units within a complex system of which compartments are the lowest level. Compartments can be seen as that allegorical nucleotide in a complex system, and they can be grouped together into a secondary layer of compositional elements called fluxes. Fluxes fill the role of codons in complex systems, and they can be grouped together to form gene-like regions within the complex system responsible for encoding specific traits.

In addition to the above, this work explored a reverse transformation on the functional decompositions using only the coefficients belonging to the eight fluxes isolated in feature selection. This reverse transformation produced a data set of adjacency matrices that could only achieve 68% accuracy in classification. This highlights a difference between the two domains that needs to be further researched.

The three fluxes isolated by the proposed methodology dimensionally reduce the feature space presented to the SVM while maintaining 100% accuracy. The classification results are used as a metric for feature validation, allowing data mining routines to search for subsets of fluxes within a system whose activity in the domain of functional decomposition optimally

differentiates system trait values. Ultimately, this methodology isolates flux groupings from a system acting as descriptors, which serve to maximally separate classes of adjacency matrices representing different states of that system.

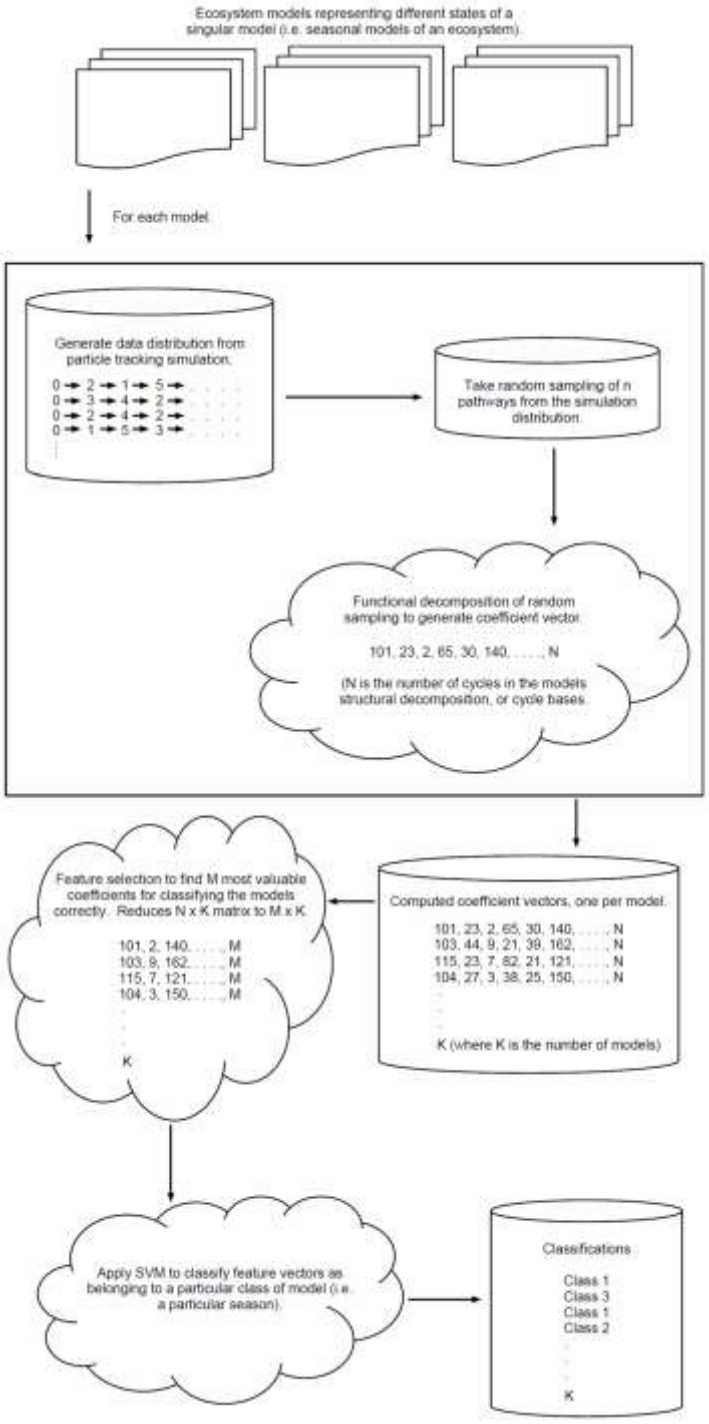


Figure 48: Flow chart of the NFA process from beginning to end.

Temporal Concept Analysis Using NFA

Temporal concept analysis is an extension of formal concept analysis (FCA) that introduces a time component to concept lattices allowing concepts to evolve. This time component establishes temporal orderings between concepts represented by directional edges connecting nodes within a temporal lattice. This type of relationship enforces a temporal link between concepts containing certain attributes. The evolution of concepts can provide insight into the underlying complex system causing change, and the concepts evolving can be seen as data emission from that complex system. This research utilizes models of complex systems to provide frequency vectors of activity in well-defined sub-networks within a system. Using NFA to analyze systems in this way can provide higher levels of contextual meaning than traditional system analysis calculations such as nodal connectedness and throughflow, thus providing unique insight into concept evolution within systems.

Introduction

FCA is a principled way of deriving ontological structures from a set of data containing objects and attributes [106]. It establishes concepts from collections of objects exhibiting a certain group of attributes. In a database void of time, these concepts appear without change, however, in temporal concept analysis [107, 108, 109] time is taken into account and concepts can evolve to take on different meaning. As an example take a database where people are objects possessing the attributes of either young or old. If time steps are present in this database a person *p* could have entries at different time steps, *t1* and *t2*, where *p t1* is labeled young and *p t2* is labeled old. This would highlight that people objects can morph from young to old over sequential time steps. This serves to establish a temporal link from time step *t* to *t + 1* between the attributes young and old. This is a simple example where a one way transition from young to

old occurs, but temporal relationships between attributes can be far more complex involving a sophisticated network of transitions and complex systems of interaction. The underlying system causing system evolution can be modeled in an adjacency matrix of transition probabilities from one attribute to another. This adjacency matrix can be seen as a kind of Markov model outlining the attribute transition probabilities for objects in a concept lattice. A temporal concept attribute model (TCAM) is a modeling of a complex system where nodes in the system are attributes and flows in the system are probabilities that objects possessing an attribute at time *t* will possess another attribute at time $t + 1$.

Temporal Concept Attribute Models (TCAM)

An attribute transition model can be constructed from a time stamped database by isolating all instances of object transition from one attribute to another over a given window of time in the database. Strategies for constructing this model include, but are not limited to, the following method: First a group of attributes *A* must be defined where *A* contains all the attributes being modeled in the TCAM over a defined time window *T*. For completeness *A* may need a null attribute value representing an object having an attribute in the model at time step *t* and then having no attribute from the model at time step $t + 1$. Once *A* and *T* are defined a set of objects *O* must be assembled that will be used to build the TCAM. *O* can be any logical grouping of objects. With *A*, *T* and *O* defined all entries it the database for each object in *O* over the time window *T* must be enumerated in time step order. For any object being labeled with attribute a_1 at time step *t* and a_2 at time step $t + 1$ a frequency of occurrence value for the edge between a_1 and a_2 in the TCAM is incremented by one. In this example we are seeking only transitions and constrain the methodology by saying an attribute is not permitted to have an edge looping back to itself. If an object stays in possession of a particular attribute for multiple time

steps nothing is modified in the transition matrix. Once every time step in *T* for every object in *O* is enumerated the transition matrix can be normalized to reflect the probability of transition between attributes. As an important note the set of attributes in *A* must never appear in any combination at the same time step for a single object. If attributes a_1 and a_2 both appear at time step *t* for object *o* a new element must be added to *A* called *a'*. This new element represents the occurrence of both attributes at the same time step. This maintains consistency in *A* such that elements of *A* are attribute state groupings that objects transition in an out of.

Discussion

Decomposing a TCAM and computing coefficients for its sub-networks (NFA) is a novel way to analyze complex systems behind concept evolution. However, work is needed to determine useful ways of applying this methodology. Using the information for data mining holds potential to be transformative and is an interesting way of allowing machines to understand concept evolution. One interesting usage of this methodology would be to compare different windows of time within a time step database using the computed vector in a distance metric. Complex systems in different states (*i.e.* they are modeled with different transition probability matrices) would have different coefficients, and the Euclidean distance between those coefficients in a particular feature space could be an invaluable source of information for data mining and knowledge extraction.

The coefficients computed by this methodology hold more information than the transition probabilities alone because they apply to higher order relationships in the system rather than simple binary relationships. Research needs to be devoted to finding ways to exploit the additional information embedded in this representation of system activity.

Conclusion

A TCAM is transition probability matrix that models attribute transition within temporal concept analysis. This work has shown how to construct a TCAM from a time stamp database. A TCAM models a complex system driving concept evolution within temporal concept analysis. NFA can be used to compute magnitude coefficient values detailing the frequency of occurrence for each sub-network in simulated data from the TCAM. Analyzing system activity with NFA allows new information about the system to be ascertained specifically related to its decomposed set of sub-networks.

This methodology has great potential, and the diverse range of problems to which it can be applied is a major strength of the work. It can be transformative for systems analysis because it provides a new domain in which system activity can be viewed.

Flux Shift Analysis of Lake Turkana

When functionally decomposing a complex system, ambiguities in the system make the solution space of the resulting coefficient vector larger than a singular point. An averaging method combining every interpretation for a pathway can be used to provide a single point of interest within this solution space, but the Difference Vector Matrix (DVM) can provide an exact bound on the hyperplane of solutions for a given vector.

Functional decompositions of structurally similar systems will yield either the same hyperplane or a disjoint one within the same feature space. Flux Shift Analysis (FSA) measures the unique orthogonal distance between the calculated hyperplanes, which is a constant distance between the functionally different systems based on cycle activity. FSA can be used for system comparison and impact analysis to understand which relationships in an ecosystem are contributing maximally to the system changing.

Introduction

Functionally decomposing a complex system involves analyzing a distribution of network pathways to interpret them as a combination of cycles rather than a sequence of nodes. As a result of inherent ambiguities, some pathways can be interpreted as multiple combinations of cycles. This makes the vector calculated by functional decomposition part of a solution space of valid vectors. The ambiguities in the system that cause this solution space can be isolated to bound it. The ambiguities are equality constraints over the coefficient vector that when combined with the inequality constraint that every coefficient value must be greater than or equal to zero (cycles cannot have negative flows) defines a hyperplane in an n dimensional feature space where n is the length of the functionally decomposed coefficient vector. This bounding set of equalities and inequalities is defined by the structure of a system, and two structurally equivalent systems will have either disjoint or equivalent hyperplanes within the same feature space. The distance between these hyperplanes can be used to analyze system change over specific cycles within the systems. Cycles are thought to hold important roles within a system [122], and this methodology calculates a unique and constant distance between cycles of structurally equivalent systems having different flow parameters. The difference in flow parameters is seen as the result of some impact or perturbation to the system and this methodology allows for the analysis of the difference throughout the exhaustive set of simple cycles in the system.

This paper will analyze a paper from Kolding [123] comparing structurally equivalent models of an ecosystem that have different flow values, apply this methodology, and compare the results in that paper with the results found herein. First, a review of flow decomposition and the DVM will be outlined. Next the Lake Turkana ecosystem will be briefly introduced and the

results of running the methodology will be shown. Finally, comparative analysis between the two snapshots of the ecosystem will be presented.

Flow Decomposition

Flow decomposition on complex systems is the process of calculating a magnitude coefficient for every simple cycle in a system, representing the amount of total system flow for which each of the cycles is responsible. These magnitude values for cycles place the system in a numeric feature space, allowing rich analysis of system activity. Luper *et al.* [87] outlined a methodology for functionally decomposing system throughflow. This methodology transformed a structurally decomposed system into a context-free grammar and used the Cocke Younger Kasami (CYK) [90, 91, 92] algorithm to calculate magnitude coefficients by parsing individual pathways from a data distribution. The CYK algorithm parsed each pathway from a system's data distribution to label each step of the pathway as belonging to a specific cycle (grammar unit). The frequency of occurrence for each cycle was tracked throughout a data distribution and stored in a vector. This vector was the basis for the magnitude values sought in the functional decomposition. Luper *et al.* [87] used an averaging technique to deal with ambiguities in pathways where every possible interpretation for a pathway was found and then weighted to provide a single localized interpretation vector used for maintaining a vector of coefficient values over the entire data distribution of pathways. This methodology provided a singular point of interest within the hyperplane of the functional decomposition, but the work herein will not use this averaging technique.

In this work only one interpretation for each pathway is needed for inclusion in the vector. The goal of this work is to find a bounding hyperplane for the functional decomposition. System ambiguities are isolated and formed into equality constraints which can be used to

transform a single interpretation into all other valid interpretations. The detected ambiguities, in conjunction with a single interpretation for each pathway in the system's data distribution, ensure there is no information loss when applying this methodology.

Difference Vector Matrix

While structurally unique decompositions exist for networks, functional decomposition is not guaranteed to be unique. Ambiguities occur when interpreting pathways in a system when different sets of cycles use the same edges the same number of times, as illustrated in the Figure 49. If a single path through a network can have multiple interpretations, the vector calculated from those interpretations will not define a single point within an *n* dimensional feature space. This ambiguity means the calculated vector defines a bounded hyperplane within the *n* dimensional feature space of the vector. The bounds of this hyperplane can be calculated by isolating each ambiguity within the system. These ambiguities define equality constraints where a particular grouping of coefficients from the vector subtracted from a related grouping of coefficients must equal zero. Logically the ambiguities depict scenarios where, given a certain path through a network, one grouping of cycles is attributed the interpretation which takes away value from a different grouping of coefficients that could also be used to interpret the pathway. The equality rules defined by the ambiguities are not the only source of information used to bound the hyperplane determining the valid solution space of the vector. Inequality rules can be defined to limit the vector values for each decomposed cycle in the system. The first set of inequalities governing the frequency vector is that every cycle must have a frequency count greater than or equal to 0. It is not possible for a cycle to have a negative flow value for the purposes of this methodology as this would indicate reverse flow against a defined edge within the system. The second set of inequalities stipulates that each value in the vector has to be less

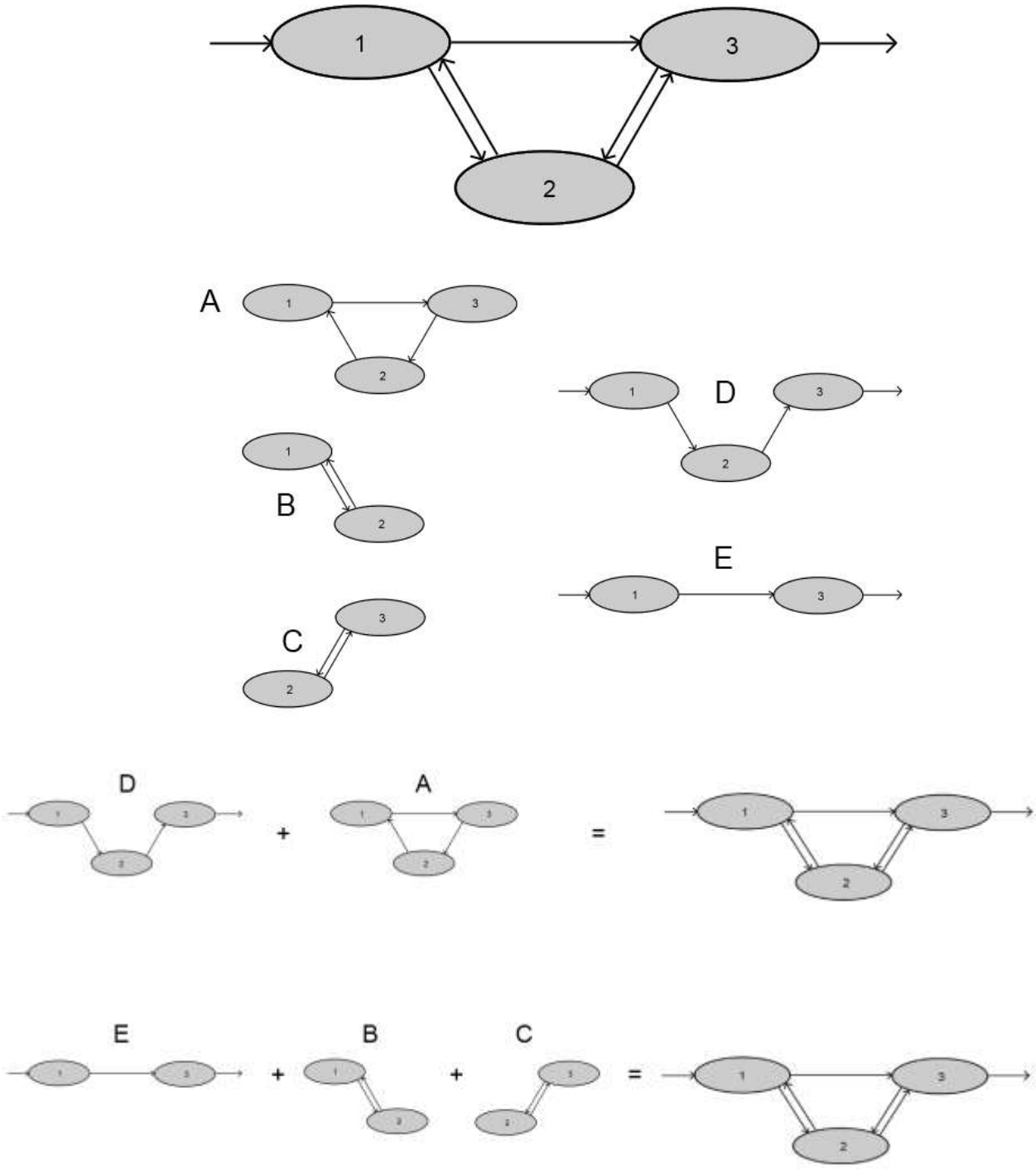


Figure 49: This figure uses a synthetic three compartment model to illustrate ambiguity that results in a Difference Vector. The system has five fluxes A, B, C, D and E. Fluxes D + A produce the complete system as does E + B + C, thus D + A = E + B + C and that equation constitutes a Difference Vector.

than or equal to the summation of all values in the vector. This set of inequalities is valid because there is a limited amount of throughflow in the system, and the vector values must reflect this conserved flow amount. The equalities and inequalities together define a hyperplane the vector must stay within.

Each pathway from the network that is parsed to calculate the frequency vector can be transformed into an interpretation vector. An interpretation vector is a local vector built from a single interpreted pathway. Each index in the interpretation vector corresponds to a cycle, and the value at each index is the number of times that cycle was used in the interpretation. An ambiguity in the system introduces the potential for certain pathways to have multiple interpretation vectors.

A Difference Vector is the result of subtracting interpretation vectors parsed from the same pathway. The Difference Vectors for a system are stored in a matrix called the Difference Vector Matrix (DVM). This matrix is the set of all linearly independent Difference Vectors in the system. The DVM can be found by performing an exhaustive combinatorial search on a structurally decomposed network. In order to find every Difference Vector, the search needs to generate two sets of combinations of cycles from the decomposed network and see if their respective multisets of edges are equivalent. If a group of cycles can be merged to form the same multiset of edges as another group of cycles, then the two groups represent an ambiguity in the network.

The structure of the search algorithm is as follows. First a network must be structurally decomposed into a set of all unique simple cycles. Let M equal the number of cycles in the decomposed network. Any ambiguity takes the following form; a grouping of at least two cycles forms an equivalent multiset of edges as a different grouping of at least two cycles. Therefore

any ambiguity requires at least four cycles. Where *m* is the size of *M* and *n* is greater than or equal to four and less than or equal to *m*, a search must be run of all possible combinations of *m* choose *n* cycles. Every generated combination of cycles is then divided into every combination of two groupings. The multisets of edges from the two groupings are compared, and if they are the same then a Difference Vector has been found. Any Difference Vector is added to the DVM if it increases the rank of the DVM. Difference Vectors that do not increase the rank of the DVM are different representations of a Difference Vector that has already been found. This search strategy finds the most concise representation of any Difference Vector because it moves through the search space starting with the shortest possible combination of cycles outward toward the longer possibilities. The runtime for this algorithm can be seen in Formula 6.

$$\sum_{m(m!/i!(m-i)!)}^{i=4} * \sum_{i(i!/j!(i-j)!)}^{j=2}$$

Formula 6: The runtime for the exhaustive search to find all Difference Vectors.

More research is needed to see if this runtime can be reduced. One way to reduce the runtime could be to establish an upper bound on the maximum length of a combination of cycles it takes to represent every Difference Vector in a search space. Another option for speeding up the search would be to compute the problem within a distributed computing framework. The search space could be partitioned and computed in parallel. Any Difference Vector that is found could be reported back to a process that manages the linearly independent matrix of Difference

Vectors. This problem can equate to the problem of finding ambiguities in a context-free grammar.

The DVM is found computationally, but it has a mathematical counterpart. If a network is decomposed and all cycles are found, a flow matrix can be made. This flow matrix has *n* columns and *m* rows, where *n* is the number of decomposed cycles *m* is the number of edges in the network. Each column of the flow matrix has a one in any row corresponding to an edge in the respective cycle and zeroes at rows corresponding to edges not used in the cycle. The null space of this matrix is related to the DVM. The DVM is a subset of the null space. Any Difference Vector in the DVM will be in the null space in some representation.

The null space can be calculated much faster than the search algorithm can computationally derive the DVM, but this null space has two problems. First the null space contains more elements than the DVM. The reason for this is there are theoretical Difference Vectors that cannot exist legally within a real system. These theoretical Difference Vectors violate a rule pertaining to the system input and output node referred to as the environment. The environment is a starting and stopping node for every pathway through the system, thus any pathway can only ever use one cycle containing this environment node. These illegal Difference Vectors use two cycles that contain the environment on at least one side of the equality constraint. The computational methodology can ignore these theoretical impossibilities, but there is no way for the null space calculation to isolate these particular Difference Vectors. If not for the second problem the first problem would be solvable. The second problem is that while the null space contains every Difference Vector, the representation of the Difference Vectors in the null space makes it impossible to tell which combination of cycles a particular equality constraint represents. The null space contains theoretical Difference Vectors (including

Difference Vectors comprised of edges with negative flow) which have multiple environmental cycles as well as representations of Difference Vectors that are not in the most concise form (*i.e.* a Difference Vector representation in the null space can have more cycles than are needed to define a particular rule). While it is theoretically possible to manipulate the null space to an equivalent format as the DVM, the methodology for this is not known.

Once a DVM is calculated it can be used to bound the hyperplane of valid frequency vector representations. The DVM is defined entirely by the structure of the graph, and thus it follows that structurally equivalent graphs have the same DVM. This property can be exploited to produce a distance value between vectors resulting from structurally similar graphs with different flow values.

For any deconstructed network, the set of equality constraints represented in the DVM combined with the inequality constraints will define a hyperplane that is disjoint from any other structurally equivalent network with different flow values. Any point on one of these hyperplanes would have a corresponding point of minimum distance on the other hyperplane. These hyperplanes are parallel so the minimum distance would be the same for any point on the hyperplane. This distance value can be used as a basis for any number of machine learning and computational intelligence methodologies to compare networks in Euclidean space. Equally as import as a single distance value, is a vector comprised of the absolute value of p2 – `p1, where p2 is a point on one hyperplane and `p1 is the corresponding point of minimum distance on the other hyperplane. This vector is called the Constant Distance Vector (CDV) and contains constant differences between the two systems for each individual cycle.

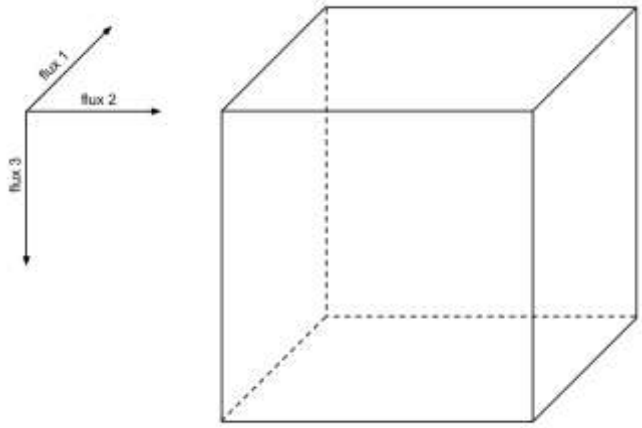


Figure 50: This figure represents a numeric feature space defined by a theoretical system with only three fluxes. A functional decomposition of a system with this structure would define a point in this feature space where the coefficient at flux 2 would be the x axis value, the coefficient at flux 3 would be the y axis value and the coefficient at flux 1 would be the z axis value.

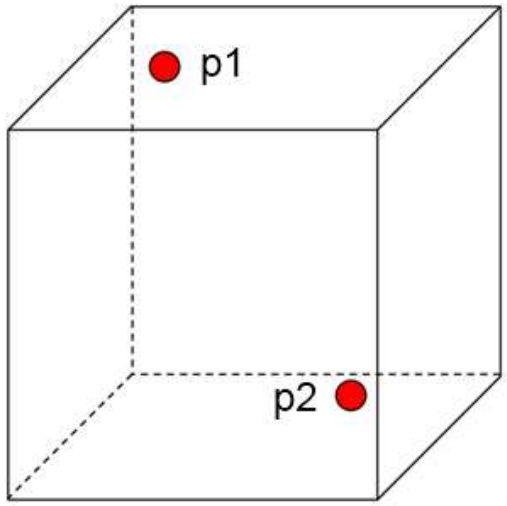


Figure 51: This figure shows the points for two functional decompositions in the feature space created from the theoretical system with three fluxes. The functional decomposition of system 1 is plotted as p1 and the functional decomposition of system 2 is plotted at p2.

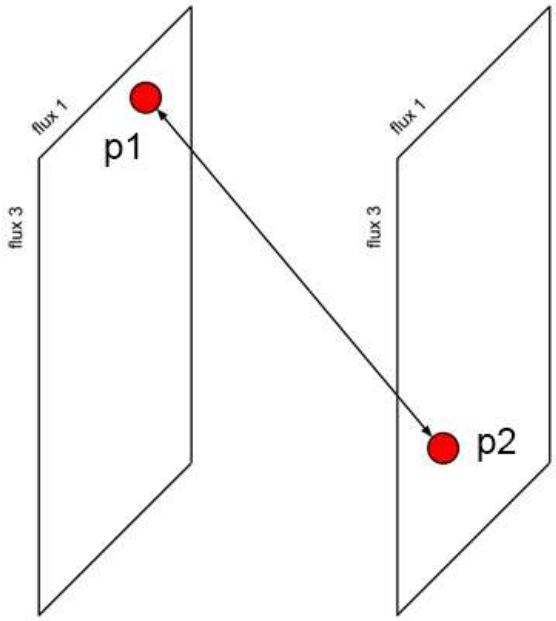


Figure 52: This figure depicts what happens when fluxes are involved in Difference Vectors. Difference Vectors define planes on which the point of a functional decomposition can be moved. The axis defined by flux 1 and the axis defined by flux 3 are degrees of freedom for the point.

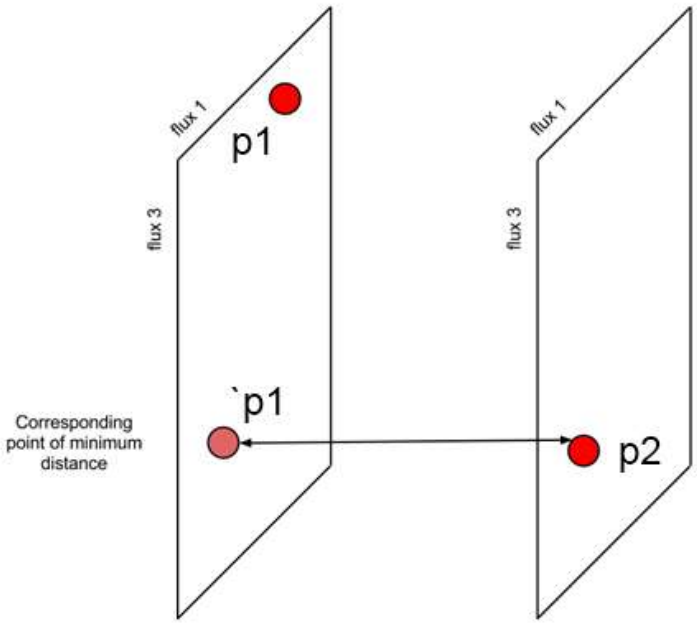


Figure 53: This figure illustrates finding the point of minimum distance for p2. This distance is the distance between the hyperplanes defined by the functional decompositions for systems 1 and 2 and by the DVM. It is a constant distance indeterminate of which p2 or p1 is chosen, and it is a unique, resolute value.

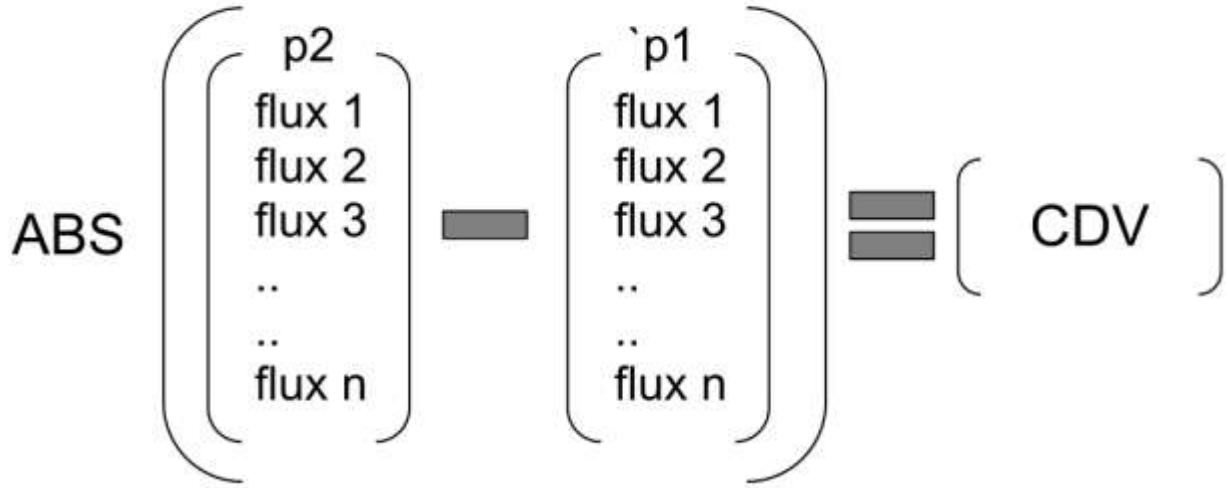


Figure 54: This figure illustrates how to obtain the Constant Distance Vector (CDV) using p2 and `p1. The CDV contains constant differences between the two systems for each individual flux.

Lake Turkana

Lake Turkana is a large body of water that sits in the North West region of Kenya in the Rift Valley. Kolding [123] analyzed two different time periods for this lake fourteen years apart using a systems analysis tool called ECOPATH II. In his paper he outlined two different steady state models for the ecosystem in the years 1973 and 1987. These models consisted of eight compartments and the environment that exchanged biomass. He used the system models to analyze the two different time periods in the ecosystem. Kolding chose these two time periods because they were the only two periods for which the data to support the models existed. The models included only important open water fish species, and some with similar biology were grouped together. These models assumed that biomass was constant, thus total production of all compartments was equal to losses. The data for the 1973 model came from Bayley [124] and Hopson [125], and all data for the 1987 model came from Kallqvist *et al.* [126] and Kolding [127].

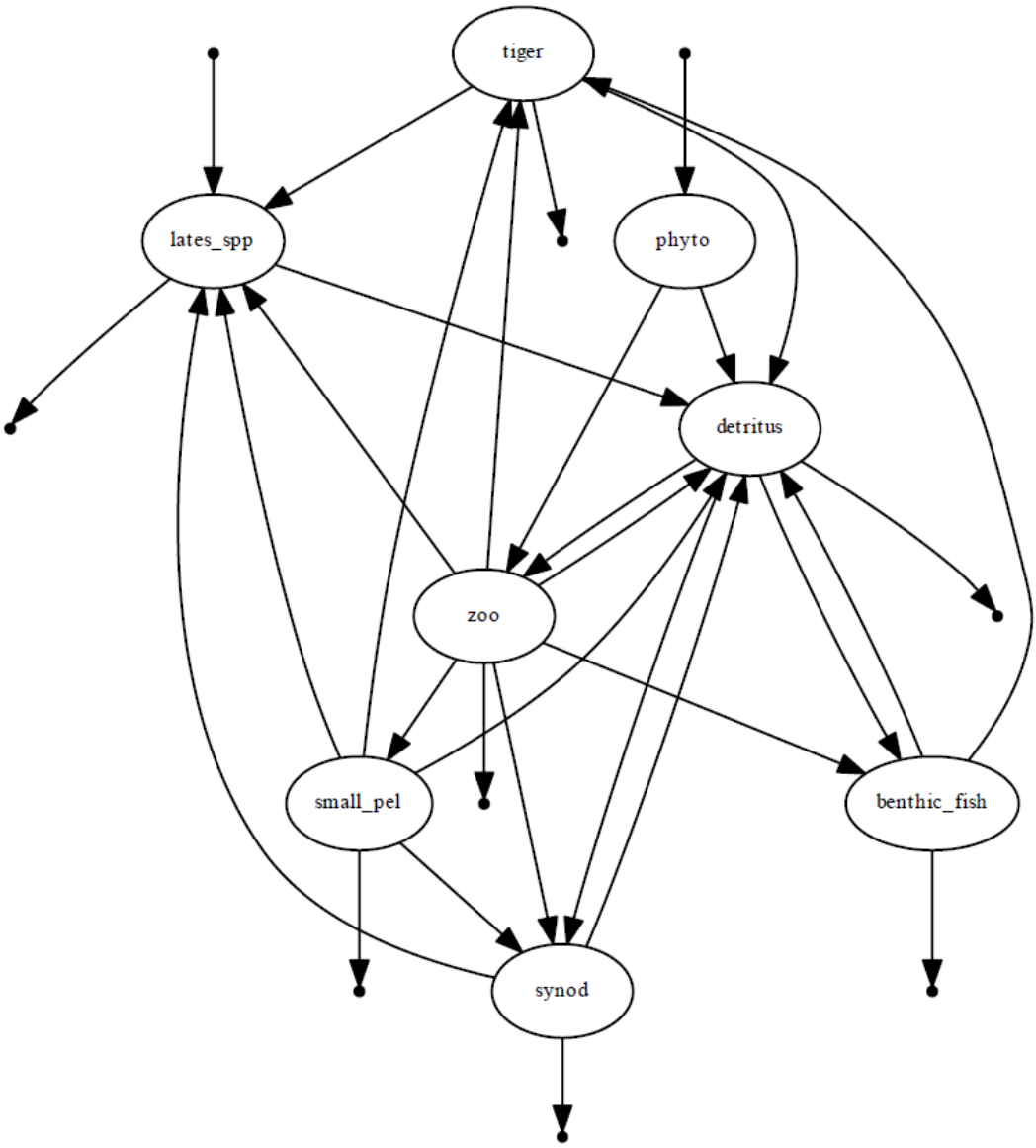


Figure 55: This figure shows the structure of the Lake Turkana ecosystem as a directed graph.

In this work Flux Shift Analysis is used to analyze change in the Lake Turkana ecosystem between the two models of Lake Turkana and to scrutinize the results mentioned by Kolding [123]. The results of running the methodology can be seen in Figure 57 and Table 11.

The FSA results are both ecologically compelling and revealing. Reference Table 11, which depicts the 5 sequences (4 chains and 1 cycle) that uniquely remain the 5 largest fractions of TST (fluxes) for both years despite the significant changes to Lake Turkana's trophic level control. First, in both the 1973 and the 1987 models, the most important cycle or chain is the zooplankton-detritus cycle representing 38.6% and 36.8% of the total system throughflow, respectively. Although control completely shifted bottom-up to top-down, given the probability of a potential 125 cycles and chains, this single cycle decisively remained the most important. Detritus and zooplankton clearly play a significant role despite drastic changes to the trophic structure of the lake ecosystem. In particular, the understanding of the importance of detritus in ecological modeling has been growing [130, 133, 134] with these results providing empirical evidence of this necessity. Clearly, detritus is important to ecological modeling strategy and perhaps zooplankton is more important than has been previously revealed.

Second, the largest flux change, representing 34.1% of the total change of TST between 1973 and 1987, occurs with the flows along the chain from env-phyto-det-env. With bottom-up control, fractions of TST along the top four most active chains were env-phyto-det-env (10.6%), env-phyto-zoo-env (17.0%), env-phyto-det-zoo-env (23.2%), and env-phyto-det-zoo-small-pel-env (2.9%). However, with top-down control, the last three of these fluxes decreased (17.0→10.5%, 23.2→19.2%, and 2.9→1.1%) in relatively significant magnitudes representing a repression of flow from lower trophic levels in from the environment through phytoplankton ultimately feeding higher trophic levels in zooplankton and beyond. Coupled with these

73 Model		83 Model	
* -> Lates_spp	c=1.4	* -> Lates_spp	c=2.1
* -> Phyto	c=7938.8	* -> Phyto	c=5566.3
Tiger_Fish -> *	c=4.1	Tiger_Fish -> *	c=0.2
Tiger_Fish -> Lates_spp	c=0.4	Tiger_Fish -> Lates_spp	c=0.2
Tiger_Fish -> Detritus	c=2.3	Tiger_Fish -> Detritus	c=0.1
Lates_spp -> *	c=4.3	Lates_spp -> *	c=6.5
Lates_spp -> Detritus	c=2.5	Lates_spp -> Detritus	c=3.1
Small_Pel -> *	c=672.7	Small_Pel -> *	c=135.8
Small_Pel -> Tiger_Fish	c=6.6	Small_Pel -> Tiger_Fish	c=0.4
Small_Pel -> Lates_spp	c=4.3	Small_Pel -> Lates_spp	c=6.3
Small_Pel -> Synod	c=6.1	Small_Pel -> Synod	c=5.6
Small_Pel -> Detritus	c=334.8	Small_Pel -> Detritus	c=58.8
Synod -> *	c=19.7	Synod -> *	c=18.2
Synod -> Lates_spp	c=0.1	Synod -> Lates_spp	c=0.1
Synod -> Detritus	c=10.5	Synod -> Detritus	c=9.7
Zoo -> *	c=5400	Zoo -> *	c=2340
Zoo -> Tiger_Fish	c=0.1	Zoo -> Tiger_Fish	c=0.0
Zoo -> Lates_spp	c=0.6	Zoo -> Lates_spp	c=0.9
Zoo -> Small_Pel	c=1024.5	Zoo -> Small_Pel	c=206.9
Zoo -> Synod	c=21.2	Zoo -> Synod	c=19.6
Zoo -> Benthic_Fish	c=0.8	Zoo -> Benthic_Fish	c=0.6
Zoo -> Detritus	c=5552.9	Zoo -> Detritus	c=2632
Phyto -> Zoo	c=3000.1	Phyto -> Zoo	c=1300
Phyto -> Detritus	c=4938.7	Phyto -> Detritus	c=4266.3
Benthic_Fish -> *	c=2.9	Benthic_Fish -> *	c=2.1
Benthic_Fish -> Tiger_Fish	c=0.1	Benthic_Fish -> Tiger_Fish	c=0.1
Benthic_Fish -> Detritus	c=1.1	Benthic_Fish -> Detritus	c=0.9
Detritus -> *	c=1836.5	Detritus -> *	c=3065.6
Detritus -> Synod	c=3	Detritus -> Synod	c=2.8
Detritus -> Zoo	c=9000	Detritus -> Zoo	c=3900
Detritus -> Benthic_Fish	c=3.3	Detritus -> Benthic_Fish	c=2.5

Figure 56: This figure shows the flow values used in for the Lake Turkana model for the 1973 and the 1987 models respectively.

decreases, the percentage of TST along the chain env-phyto-detritus-env increased dramatically (10.6→33.7%). That is, flow from the environment into phytoplankton, rather than moving up to higher trophic levels, simply moves into detritus and then exits the system back to the environment. Considering the hierarchal level of importance granted these four flows (of the total 125 cycles and chains), this appears to be an important addition of empirical evidence confirming Kolding's assertion that control shifted from bottom-up to top-down between the years 1973 and 1987.

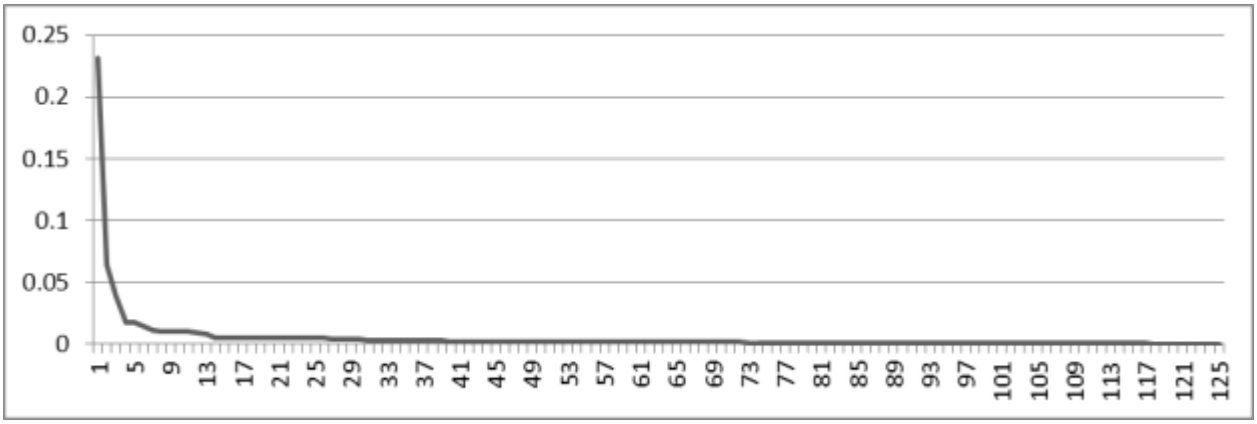


Figure 57: This graph shows the distance values for the fluxes in the Lake Turkana ecosystem plotted on a line graph sorted in descending order. The top 6 fluxes on this graph (top 5% of the fluxes) account for 57% of the distance between the two models.

Table 11: Five largest flux sequences from 125 total as generated from a Network Flux Decomposition of Lake Turkana (Kolding [123]).

Table 11, Five largest flux sequences from 125 total as generated from a Network Flux Decomposition of Lake Turkana (Kolding [123])			
Chain or Cycle	yr & control	% of TST	% contribution of total change between years
Env→Phyto→Detritus→Env (chain)	1973 bottom-up	10.6%	**34.1%**
	1987 top-down	**33.7%**	
Env→Phyto→Zoo→Env (chain)	1973 bottom-up	17.0%	9.6%
	1987 top-down	10.5%	
Env→Phyto→Detritus→Zoo→Env (chain)	1973 bottom-up	23.2%	5.9%
	1987 top-down	19.2%	
Env→Phyto→Detritus→Zoo→Small→Pel→Env (chain)	1973 bottom-up	2.9%	2.5%
	1987 top-down	1.1%	
Zoo↔Detritus (cycle)	1973 bottom-up	**38.6%**	2.5%
	1987 top-down	**36.8%**	

Conclusion

This work has presented a new methodology that can be used to compare snapshots of an ecosystem, where the snapshots have differing flow values. This methodology also provides a way to ascertain which cycles are most responsible for the differing states between the two versions of the models. The Lake Turkana ecosystem in Kenya, Africa was analyzed, specifically comparing results obtained from this methodology to those obtained by Kolding [123]. Significant ecological inference was made; illustrating this methodology allows powerful analysis of ecosystems in ways not currently available through other methodologies. This paper also outlines the computational procedures of the methodology and discusses limitations and further research that could be directed at improving the scalability of the methodology.

CHAPTER 5

SUMMARY AND CONCLUSION

Complex systems are everywhere. As humans we model what we do not understand into constructs that allow us to gain traction towards their comprehension. This tendency is seen throughout scientific disciplines yet the totality of our understanding about the construct we choose is limited. Complex systems exude characteristics seen in higher life forms such as self-organization, emergent behaviors, and evolution; however little is known about what drives these characteristics. Currently, there is no agreed-upon method for tracing a high-level abstraction such as a system trait or behavior to the compositional elements in a system responsible for encoding the information. Without the ability to dissect a system in this way, progress towards truly understanding complexity within systems is difficult. In the absence of this kind of methodology there is simply too much to compute, and observational analysis gets lost in an endless sea of information.

The methodology formulated in this paper has created an encoding framework for system activity that can be used in certain situations to increase the accuracy of machine learning algorithms and to extract encoding regions within complex systems responsible for emergent behaviors, self-organization and system traits. At the core of this framework two main concepts were introduced in the methodology section of this work.

- Structural decomposition of a complex system into compositional units called fluxes. These fluxes compose an exhaustive set of fundamental relationships in

the system, and the activation values of these relationships encode information about the system.

- Functional decomposition of a structurally decomposed system to calculate the activation for each flux in the system.

The research described here utilizes a computational methodology to search an adjacency matrix representation of a complex system in order to find every simple cycle in the system. These cycles are referred to as fluxes, and activation values are computed for these fluxes using a computational methodology called functional decomposition. Functional decomposition analyzes a distribution of pathways taken through the system by simulated particles and parses these pathways using the exhaustive set of fluxes as a context-free grammar able to interpret any pathway through the system. The activation values for each flux are calculated by interpreting this simulated distribution and tallying the frequency with which a flux appears in the parse trees of the pathways therein.

In the application chapter the concepts above are used in various ways. First, they are used to create a new analytic methodology called Network Flux Analysis (NFA). NFA uses flux activation values as input to machine learning and feature selection algorithms. NFA isolates flux sets within a complex system that encode system traits such as seasonality (*i.e.* fall, winter, spring, summer). These flux sets are encoding regions that can be used to reliably infer the value of the system trait being sought. In addition to knowledge extraction NFA has been shown to increase classification accuracy in certain situations. An application of NFA was used to boost the ability to classify which year a data distribution came from in the Neuse River estuary. NFA

boosted the performance for this classification 37.5% in comparison to using the edges in the adjacency matrix as feature input.

Another way structural and functional decomposition have been used is to perform an impact analysis study on the Lake Turkana ecosystem in the Kenyan Rift Valley. In this application a construct called the Difference Vector Matrix (DVM) is used to compute a resolute, constant distance for each flux in the ecosystem between models that are fourteen years apart. The distance a flux has moved in the fourteen-year period between the two models is used to analyze change in the ecosystem over that fourteen year period.

It is the position of this work that data instances, in any form, are sampling points or emissions from an underlying complex system. Understanding the behaviors, structure and traits of this underlying system gives a computational edge towards data analysis. Nowhere is this edge more important that in today's world of Big Data, where incredibly large, heterogeneous data streams are constantly being mined for information. The methodology proposed in this paper presents a framework through which the system beneath the data can be understood with clarity.

This methodology has implications for data mining, knowledge extraction, Big Data, computational intelligence, artificial intelligence, and many more areas from other disciplines. The general applicability of this methodology towards numerous disciplines and problems stems from the weighted adjacency matrix construct at the core of the methodology, which takes weighted adjacency matrices and transforms them into a vector in a new domain of functional decomposition. Weighted adjacency matrices appear across numerous disciplines and are commonly used to represent weighted directed graphs. Graphs are a common way to illustrate complex systems and thus saturate academia throughout numerous scientific disciplines. Any

data set that can be described using a graph can be analyzed with the methodology proposed herein.

One example of a possible application of the methodology is Markov models, which are used throughout numerous disciplines to describe phenomena (Ashbrook and Starner [115] used Markov models to predict user locations from logged GPS data). Markov models are easily representable as weighted adjacency matrices, and applying the methodology outlined herein could help boost the accuracy of location predictions. If the data were modeled according to day of the week, fluxes from the proposed methodology would represent location sequence cycles and the activations at the fluxes would represent probability of occurrence. These fluxes could help understand which patterns of movement are differentiating days of the week, providing insight into the underlying complex system driving movement in certain patterns.

Currently, activity within complex systems is studied through metrics such as edge and node analysis. Understanding system activity through edges is not ideal because edges are binary relationships in systems with higher order relationships. Flow moving across an edge can be coming and going to many different nodes within a graph, and a binary relationship does not account for this dispersion. The concept of fluxes introduced by NFA defines these higher order relationships and is able to utilize them for a new level of systems analysis more powerful than analysis of simple binary relationships.

One reason fluxes have so much potential for data analysis is that the activation values at fluxes are a result of probability distributions throughout the entire system. While a unit of flow traverses a flux it has a probability of exiting the flux at each new node it traverses. This probability is defined by the connecting edge flows for that node. If an edge is changed it can change the activation of numerous fluxes in the system, even fluxes that don't contain the edge,

due to the shift in the probability that particles in the system will traverse the edge. It is this sensitivity that gives power to the analysis of fluxes. Analyzing these higher order relationships gives greater resolution to the study of complex systems than merely viewing them through analysis of binary relationships.

Dealing with heterogeneous data is currently an active topic of research and necessary for knowledge extraction and Big Data analysis. A major benefit of analyzing complex system through the use of fluxes is that fluxes are independent, fully contained sub-systems within the overall system. Fluxes can be studied independently from the system presenting the ability to compare different complex systems that have similar relationships. This provides the ability for comparative analysis and knowledge extraction to graphs whose overall structure is not the same but have certain overlapping substructure. An example of this application would be the analysis of two different ecosystems in a manner similar to the Neuse River estuary analysis in the application section of this work. Such an analysis would be possible if the ecosystems shared common compartments and relationships. For instance, if another ecosystem had the following compartments present in the Neuse River estuary model PN_hetero, NH4, NOx, and N_sed, it would be possible to provide comparative analysis of any flux that appeared in both systems, even though the systems would be considered heterogeneous data instances. Possibilities for analysis of relationships across structurally different complex systems and the ability to perform knowledge extraction in heterogeneous environments are attractive benefits of the proposed methodology.

Current Limitations and Further Research

The methodology presented holds many avenues for further research as well as limitations that need to be addressed. The following addresses both topics.

Limitations

One of the main limitations of the work presented herein is scalability. Functional decomposition relies on the CYK algorithm and as such has a runtime of $O(n^3)$; however, the biggest consideration towards runtime is the size of the adjacency matrices being functionally decomposed. The search method involved in computing the structural decomposition of a system must find every simple cycle in the adjacency matrix. The runtime for this search is $O(|V| + |E|)$, growing in proportion to both the number of vertices and the number of edges in the adjacency matrix. This work has shown applications of the proposed methodology on real world data so the runtime limitations mentioned are not crippling; however care must be taken when building adjacency matrices for use in this methodology that those matrices not become incomputable for a structural decomposition.

Another limitation currently present in the proposed methodology is finding all of the Difference Vectors in an adjacency matrix. For the analysis presented on the Lake Turkana ecosystem, the Difference Vectors were found using an exhaustive search strategy that tried all possible combinations of cycles over the entire adjacency matrix. The runtime for this search strategy is factorial. This problem of finding Difference Vector is the same problem as finding ambiguity with context-free grammars. Finding ambiguities in a context-free grammar is an undecidable problem in the general case; however, finding ambiguities within NFA is a special case for that problem because the ambiguities being sought are in a reduced form where a cycle can appear only one time within an ambiguity (*i.e.* Difference Vector). This constraint makes the search space finite and computable, however large it may be.

One more limitation in the proposed methodology is that when extracting fluxes encoding a particular system trait there is no guarantee the best solution is going to be found. This common in machine learning, but should always be mentioned.

Further Research

One of the particularly exciting aspects of the work herein is the numerous paths it creates for further research. This works presents an entirely new framework with which to analyze complex system activity that has potential for far reaching impact throughout numerous disciplines.

Scalability

Scalability is one area where further research is needed. One idea for dealing with the limitations posed scalability in this methodology is to contract larger models in to smaller ones. Edge contraction removes edges from a graph and merges the vertices on either end of the edge. This process may not be ideal from a modeling standpoint but it is likely Network Flux Analysis (NFA) on contracted models would produce beneficial results; in effect, a contracted model would present a lower resolution view of system activity to NFA. Graph contraction could be done by domain experts in a strategized manner; however, it is also plausible that a search strategy such a genetic algorithm or particle swarm optimization could randomly search for optimal way to contract a graph for a specific problem.

When contracting an edge fluxes containing that edge ultimately wind up being merged as well. Strategic contraction of the model could leave crucial fluxes intact while merging non-crucial fluxes. In the example presented involving analysis of the Neuse River estuary, knowledge pertaining to seasonality is extracted. There are certain fluxes in this model that are more important than other to understanding seasonality, and a search strategy such as a genetic

algorithm or particle swarm optimization could find optimal ways of contracting models that leave important fluxes intact.

Another potential venue for further research towards scalability is to look into the effects of cutting systems into smaller sub-systems and applying NFA collectively to those smaller systems rather than to the entire system as a whole. It is likely that applying NFA to smaller sub-systems would yield actionable information about the activity of the entire system as a whole.

Further research needs to be invested into finding a more efficient way to isolate Difference Vectors within a system. One possible approach is finding Difference Vectors as they occur when parsing pathways. Difference Vectors are ambiguities in the context-free grammar being used to parse pathways; and as such, when multiple parse trees are generated there is an ambiguity present in the pathway being analyzed. This ambiguity is easy to find by the following methodology.

- Using two parse trees for a particular pathway, create interpretation vectors for each of them where an interpretation vector is a vector whose indexes relate to grammar elements used to parse the pathway.
- Each grammar element has an index in the interpretation vector and the value at each index indicates the number of times the grammar element was used in the interpretation.
- Subtract these two interpretation vectors and the result is a Difference Vector.

This methodology can find all Difference Vectors that occur in the distribution of pathways being analyzed, but if an instance of a Difference Vector is not present in the

distribution, then it will not be found. This means that finding all Difference Vectors is not guaranteed using this methodology; however, fluxes have a probability of occurring in combination within a data instance, which means Difference Vectors also have a probability of occurring. If enough pathways are parsed, statistically probable Difference Vectors will be found. Research should be undertaken to explore whether finding statistically probable Difference Vectors is good enough, as opposed to a methodology that is guaranteed to find all Difference Vectors (even ones with a low probability of occurrence). In any event, the methodology used for finding Difference Vectors in this work was an exhaustive, unintelligent search which should leave room for easy enhancement in the event enhancement is possible.

New Data Sets

An obvious area for further research is to find new and exciting ways to apply the methodology presented. In this work, the Neuse River estuary was analyzed and knowledge extraction was performed on seasonality traits within the system. The methodology was also applied to the Lake Turkana ecosystem for the purposes of impact analysis. The generic nature of this methodology's application toward adjacency matrices holds promise for its applicability in many other scientific areas such as economics, biological systems modeling, and social network mining.

Parallelization

As computers grow in power, more processors are becoming available, allowing problems to be computed by massively parallel systems. The methodology presented in this work could greatly benefit from parallel computation. At the core of the problem is building parse trees for simulated pathways through a system. In order to functionally decompose a system, the presented methodology builds parse trees and tracks grammar elements within the

parse trees for large distributions of pathways. Each of the pathways (which can number in the hundreds, thousands, or even greater) must be parsed in order to keep a summed tally of the frequency for grammar elements used in the pathway. This process can be done disjointly for each pathway being parsed without the need for any inter-process communication other than to provide a final report of the frequency results for each of the grammar units. The ability to parse each pathway disjointly and sum the results means a machine with 100 processors (who all have their own RAM) could functionally decompose a system 100 times faster than a machine with 1 processor, minus minimal transportation time. This performance boost would help deal with scalability constraints, as the runtime to functionally decompose a model could become more efficient by orders of magnitude.

Unordered Fluxes

The methodology in this work computes activation values for ordered sets of vertices within systems. A strategy for reducing the runtime and increasing the overall efficiency of the methodology would be to explore whether activations for unordered sets of vertices could be computed more quickly, and whether those activation values would provide comparable results to the ones presented herein. Having already computed the activation values for the ordered sets of vertices, a researcher could easily combine fluxes that, when viewed as unordered sets, are equivalent and could sum their activation values. This approach would reduce the functional decompositions presented from ordered functional decompositions to unordered functional decompositions. These unordered functional decompositions could be analyzed using similar techniques as the ones presented by this work to compare their usefulness for systems analysis.

Flux Signals in Chronological Data

The application of Network Flux Analysis towards the Neuse River estuary in North Carolina produced sixteen chronological functional decompositions. This means that each flux has a numeric time series of sixteen values. The feature selection routines performed on the 640 fluxes in the functional decomposition selected fluxes whose time series presented a more periodic function shape. Further research should be given to applying signal processing routines to the fluxes as part of the overall knowledge extraction process. The flux time series presented by the chronological functional decompositions are signals being produced by the fluxes. This concept could be helpful as signal processing is a well-established discipline with a robust set of methodologies that could easily be applied to the flux signals. It may be the case that certain signal patterns at different combinations of fluxes holds crucial system information to aide in the knowledge extraction process. In the case of extracting knowledge from the Neuse River estuary data set, the results may be enhanced be giving preference during feature selection to fluxes whose signal is more periodic in shape. The current methodology presented in this work did not give this preference, as no signal processing was performed to evaluate the extent to which a flux's signal was periodic.

Reverse Transformation

This work illustrated that a reverse functional decomposition can transform a functional decomposition vector back into an adjacency matrix. Further research needs to be devoted to investigating use cases for this procedure. This reverse transformation can act as band pass filter for flux activity. An example of this would be to apply the reverse transformation on a functional decomposition of the Neuse River estuary, but zeroing out all the coefficients except the three that were isolated during feature selection. These three coefficients can determine

seasonality over the data sets provided for the Neuse River estuary. If all the other coefficients are zeroed out and the reverse transformation is applied, the resultant adjacency matrix would only include the flow information for those three isolated fluxes. This process would provide a new adjacency matrix in which the flow values across the edges would be different than in the original adjacency matrix. The adjacency matrix produced by this reverse transform would allow real valued portions of edges to be removed. Analyzing how much flow for a particular edge in the original adjacency matrix was retained in the matrix produced by the reverse transform would enable one to compute the percentage of the flow across that edge which was used towards seasonal classification in the model. This could be computed for all edges giving a valuable piece of information to domain experts. This is one example of applying the reverse transform. It is likely more applications would easily come to light if further research is given to the reverse transform of a functional decomposition.

Applying NFA Toward a Metric for System Health

The health of a complex system is a nebulous concept. Without a clear definition there can be no rigid, empirically-based metric to rank system health. Network Flux Analysis holds potential for extracting health descriptors from a complex system to aid domain experts in empirically evaluating system health.

If a model of a system is observed by a domain expert it is usually the case a clear cut binary quantification of system health as *healthy* or *unhealthy* cannot be ascertained except in situations where the system may be in extreme duress; however, a domain expert could score system health on a scale from 1 to 10 based on whatever criteria they desire. If multiple instances of a single structural model (*i.e.* one model different flow values) are presented to multiple domain experts and those experts are to score system health in this way, Network Flux

Analysis could be applied in such a way as to extract fluxes from the system whose activation values are useful towards regressing to the score values presented by the domain experts. NFA used in this way could extract descriptors for overall system health, thus holding exciting ramifications for transformative knowledge extraction. Application of NFA in this way can work to quantify and empirically ground analysis of system concepts that lack rigid definitions.

REFERENCES

[1] L. A. Zadeh, Outline of a new approach to the analysis of complex system and decision processes, IEEE Trans. Syst. Man Cybern. Vol. 3 pp 28-44, 1973.

[2] Strogatz, S. H., Exploring complex networks, Nature 410, 268–276 (2001).

[3] Rocha, Luis M. (1999). "Complex Systems Modeling: Using Metaphors From Nature in Simulation and Scientific Models". BITS: Computer and Communications News. Computing, Information, and Communications Division. Los Alamos National Laboratory. November 1999.

[4] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," IEEE Trans. Comp.. vol. C-35, pp. 677-691. Aug. 1986.

[5] L. Wiskott, J. Fellous, N. Kruger, and C. von der Malsburg, ªFace Recognition by Elastic Bunch Graph Matching,º IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 19, no. 7, pp. 775-779, July 1997.

[6] Saul, L.k. and Jordan, M.I., Mixed memory Markov model: Decomposing complex stochastic processes as mixture of simpler ones, Machine Learning, Vol.37, 75-88, Oct. 1999

[7] S. Mitra, S.K. Pal, and P. Mitra, "Data Mining in Soft Computing Framework: A Survey," IEEE Trans. Neural Networks, vol. 13, pp. 3-14, 2001.

[8] Goebel, M., & Gruenwald, L. (1999). A survey of data mining and knowledge discovery software tools. ACM SIGKDD, 1(1), 20–33.

[9] S. Muthukrishnan. Data streams: Algorithms and applications. Foundations and Trends in Theoretical Computer Science, 1:117–236, 2 2005.

[10] J. Hipp *et al.*, "Data Quality Mining", DMKD2001 Workshop on Research Issues in Data Mining and Knowledge Discovery DMKD2001, 2001. *

[11] Mitchell TM: Machine Learning. WCB/McGraw-Hill 1997.

[12] R.C. Eberhart and Y. Shui, Computational Intelligence - Concepts to Implementations, Elsevier, 2007.

[13] Boulding, K.E. General Systems Theory – The Skeleton of Science. Management Science, 1956, 2, 197 – 208

[14] Ackoff, R.L. (1971). Toward a System of Systems Concepts. Management Science, 17 (11), pp. 661-671.

[15] von Bertalanffy, L. (1968). General system theory. New York: George Braziller Inc.

[16] Caldarelli, G. & Vespignani, A. (eds), Large Scale Structure and Dynamics of Complex Networks (World Scientific Press, Singapore 2007).

[17] Bossomaier 96] T.R.J. Bossomaier and D.G. Green, "Complex systems", Cambridge University Press, Amsterdam, 1996.

[18] F. Szidarovszky and A. T. Bahill, Linear Systems Theory. Boca Raton,FL: CRC, 1998.

[19] Deisboeck, T. S. and Kresh, J. Y., Complex systems science in biomedicine. New York, Springer, 2006.

[20] Sethna, J. P. Statistical Mechanics: Entropy, Order Parameters and Complexity. Clarendon Press, (2007).

[21] Prigogine, Ilya. (1945). Etude Thermodynamics des Phenomenes Irreversibles (Study of the Thermodynamics of Irreversible Phenomenon). Presented to the science faculty at the Free University of Brussels (1945); Paris: Dunod, 1947

[22] Barnsley, M.F., Fractals Everywhere, 2nd edition, Academic Press, San Diego, 1993.

[23] Prusinkiewicz, P. Graphical applications of L-systems. Proc. of Graphics Interface '89 (1986), 247-253

[24] M. Sipser. Introduction to the Theory of Computation. PWS Publishing, 1997.

[25] Ano, M, and Kunii, T. L. [1984]: Botanical tree image generation. IEEE Computer Graphics and Applications 4, Nr. 5, pp. 10-34.

[26] Smith, A. R. [1984]: Plants, fractals, and formal languages. Computer Graphics 18, Nr. 3, pp. 1-10.

[27] Anastacio, F., Prusinkiewicz, P., Sousa, M. C., Sketch-based Parameterization of L-systems using Illustration-inspired Construction Lines, Computers and Graphics, 2009, Vol. 33, Issue 4, pp. 440-451

[28] McCormack, J: 1993, Interactive Evolution of L-System Grammars for Computer Graphics Modelling, in Green, D and Bossomaier, T (eds), Complex Systems: from Biology to Computation, ISO Press, Amsterdam, pp. 118-130.

[29] Prigogine, I. (1980). From Being to Becoming. W.H. Freeman and Co, San Francisco.

[30] Prigogine, Ilya; Nicolis, G. (1977). Self-Organization in Non-Equilibrium Systems. Wiley

[31] Prigogine, Ilya; Stengers, Isabelle (1984). Order out of Chaos: Man's new dialogue with nature. Flamingo.

[32] D. G. Green and T. Bossomaier. Complex Systems: from Biology to Computation. IOS Press, 1993

[33] Green, D. G., D. Newth, and M. Kirley. (2000). Connectivity and catastrophe - towards a general theory of evolution. Pages 153-161 in M. A. Bedau, editor. Artificial Life VII: Proceedings of the Seventh International Conference. MIT Press.

[34] Pickett, S.T.A., Cadenasso, M.L., 2002. Ecosystem as a multidi- mensional concept: meaning, model and metaphor. Ecosys 5, 1–10.

[35] M. S. Chen, J. Han, and P.S. Yu. Data Mining: An Overview from a Database Perspective. IEEE Transaction on Knowledge and Data Engineering, 8(6):866-883, December 1996.

[36] Z. Chen, From Data Mining to Behavior Mining. International Journal of Information Technology & Decision Making, Vol. 5, No. 4 (2006) 703-711

[37] Studies in the History of Probability and Statistics: IX. Thomas Bayes's Essay Towards Solving a Problem in the Doctrine of Chances Author(s): G. A. Barnard and Thomas Bayes Source: Biometrika, Vol. 45, No. 3/4 (Dec., 1958), pp. 293-315 Published by: Biometrika Trust, Stable URL: http://www.jstor.org/stable/2333180

[38] Heckerman, D. Bayesian networks for data mining. Data Mining and Knowledge Discovery, submitted.

[39] J. Han, M. Kamber, Data Mining : Concepts and Techniques, Morgan Kaufmann, August 2000.

[40] Ian H. Witten and Eibe Frank. Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann, 1999.

[41] Goebel, M., & Gruenwald, L. (1999). A survey of data mining and knowledge discovery software tools. ACM SIGKDD, 1(1), 20–33.

[42] G. Piatetsky-Shapiro and W. J. Frawley. Knowledge Discovery in Database. AAAI/MIT Press, 1991.

[43] Michael W. Berry, editor. Survey of Text Mining: Clustering, Classification, and Retrieval. Springer, September 2003. ISBN: 978-0-38795-563-6.

[44] Pierre Zweigenbaum, Dina Demner-Fushman, Hong Yu, and Kevin B. Cohen. 2007. Frontiers of biomedical text mining: Current progress. Briefings in Bioinformatics.

[45] S. Mitra and T. Acharya. Data Mining: Multimedia, Soft Computing, and Bioinformatics. J. Wiley & Sons, New York, 2003.

[46] M¨uller, K.-R., Smola, A., R¨atsch, G., Sch¨olkopf, B., Kohlmorgen, J. and Vapnik, V. Predicting time series with support vector machines. In Proceedings, International Conference on Artificial Neural Networks, page 999. Springer Lecture Notes in Computer Science, 1997.

[47] Burges, C.J.C. (1998) "A tutorial on support vector machines for pattern recognition." Data Mining and Knowledge Discovery, Vol. 2(1), 121-167.

[48] R. C. Eberhart, and Y Shi. Computational Intelligence: Concepts to Implementations, Morgan Kaufmann Publishers. San Francisco, (in press).

[49] Fu, T. C., Chung, F. L., Ng, V. & Luk, R. (2001). Pattern Discovery from Stock Time Series Using Self-Organizing Maps. Workshop Notes of KDD2001 Workshop on Temporal Data Mining. San Francisco, CA, Aug 26-29. pp 27-37.

[50] X. Cai, N. Zhang, G. Venayagamoorthy, and D. Wunsch. Time series prediction with recurrent neural networks using a hybrid pso-ea algorithm. In IJCNN, pages 1647–1653, 2004.

[51] Kim, K. J. (2003). Financial time series forecasting using support vector machines. Neurocomputing, 55(1/2), 307–319.

[52] Shin, K.S., Lee, T.S., & Kim, H.J. (2005). An application of support vector machines in bankruptcy prediction model. Expert Systems with Applications, 23(3), 321-328

[53] Burbidge, R.; Trotter, M.; Buxton, B.; Holden, S. Drug design by machine learning: support vector machines for pharmaceutical data analysis. Comput. Chem. 2001, 26, 5-14.

[54] A. A. Freitas, "A survey of evolutionary algorithms for data mining and knowledge discovery," in Advances in Evolutionary Computation, A. Ghosh and S.S. Tsutsui, Eds. New York: Springer-Verlag, 2001.

[55] J.H. Holland, (1998) Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to biology, control and artificial intelligence. MIT Press, ISBN 0-262-58111- 6. (NB original printing 1975).

[56] Povinelli, 2000] Povinelli, R. J. (2000). Using Genetic Algorithms to Find Temporal Patterns Indicative of Time Series Events. Workshop on Data Mining with Evolutionary Computation held in GECCO2000, pages 80 - 84.

[57] Eads, D. R., Hill, D., Davis, S., Perkins, S. J., Ma, J., Porter, R. B., *et al.* (2002). Genetic algorithms and support vector machines for time series classification. Proceedings of SPIE, 4787, 74–85.

[58] Eberhart, R. C. and Kennedy, J. A new optimizer using particle swarm theory. Proceedings of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan. pp. 39-43, 1995

[59] Merwe V. D. and Engelbrecht, A. P., 2003. Data clustering using particle swarm optimization. Proceedings of IEEE Congress on Evolutionary Computation 2003 (CEC 2003), Canbella, Australia. pp. 215-220.

[60] Parpinelli RS, Lopes HS, Freitas AA. Data mining with an ant colony optimization algorithm. IEEE Trans Evolutionary Computation 2002;6(4):321–32.

[61] Frey BJ, Dueck D. Clustering by passing messages between data points. Science. 2007;16:972–976

[62] W. Wilson, P. Birkin and U. Aickelin, "Motif detection inspired by immune memory", 6th International Conference on Artificial Immune Systems (ICARIS), Volume 4628 of Lecture Notes in Computer Science 4628, pp. 276-287, Santos, Brazil, 2007.

[63] J. E. Hunt and D. E. Cooke, "Learning Using an Artificial Immune System", Journal of Network and Computer Applications, vol. 19, pp. 189-212, 1996.

[64] JAMES C. BEZDEK, ROBERT EHRLICH, WILLIAM FULL, "THE FUZZY c-MEANS CLUSTERING ALGORITHM", Computers & Geosciences Vol. 10, No. 2-3, pp. 191-203, 1984.

[65] X. Cai, N. Zhang, G. Venayagamoorthy, and D. Wunsch. Time series prediction with recurrent neural networks using a hybrid pso-ea algorithm. In IJCNN, pages 1647–1653, 2004.

[66] Rivas, V.M., Merelo, J.J., Castillo, P.A., Arenas, M.G., Castellano, J.G. (2003). Evolving RBF neural networks for time-series forecasting with EvRBF. Information Sciences, forthcoming.

[67] Grinsted, A., J. C. Moore, and S. Jevrejeva (2004), Application of the cross wavelet transform and wavelet coherence to geophysical time series, Nonlinear Processes Geophys., 11(5/6), 561– 566.

[68] Eads, D. R., Hill, D., Davis, S., Perkins, S. J., Ma, J., Porter, R. B., *et al.* (2002). Genetic algorithms and support vector machines for time series classification. Proceedings of SPIE, 4787, 74–85.

[69] W. Wilson, P. Birkin and U. Aickelin, “Motif detection inspired by immune memory”, 6th International Conference on Artificial Immune Systems (ICARIS), Volume 4628 of Lecture Notes in Computer Science 4628, pp. 276-287, Santos, Brazil, 2007.

[70] B Chiu, B, Keogh, E. and Lonardi, S., 2003. Probabilistic Discovery of Time Series Motifs. In proceedings of the 9th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining. Washington DC, USA, Aug 24-27. pp. 493-498. SIGKDD, August, 2003.

[71] Thomas Cormen, Thomas, Charles Leiserson, and Ronald Rivest. 2001. Introduction to Algorithms. Second Edition, Cambridge, MA: MIT Press.

[72] R. Jiang, Z. Tu, T. Chen, and F. Sun, “Network motif identification in stochastic networks,” Proc. Nat. Acad. Sci. U.S.A., vol. 103, no. 25, pp. 9404–9409, Jun. 2006.

[73] P. Ammann, D. Wijesekera, S. Kaushik, “Scalable, Graph-Based Network Vulnerability Analysis,” in Proceedings of CCS 2002: 9th ACM Conference on Computer and Communications Security, Washington, DC, November 2002.

[74] Carley, K.M., 2003. Dynamic network analysis. In: Breiger, R. Carley, K. Pattison, P. (Eds.), Dynamic Social Network Modeling and Analysis: Workshop Summary and Pappers. Committee on Human Factors, National Research Council, pp. 133-145.

[75] Subbu, A. and Ray, A. Space partitioning via Hilbert transform for symbolic time series analysis. Appl. Phys. Lett., 2008, 92(8), 084107-1–084107-3.

[76] Batagelj, V. and Mrvar, A. (1998). Pajek Program for large network analysis. Connections, 21(2):47, 57. Project home page at http://vlado.fmf.uni-lj.si/pub/networks/pajek/.

[77] Smith, David A., and Douglas R. White. 1992. "Structure and Dynamics of the Global Economy: Network Analysis of International Trade, 1965-1980." Social Forces 70:857-93.

[78] Wellman B, Salaff J, Dimitrova D, Garton L, Gulia M, Haythronwaite C. 1996. Computer networks as social networks: collaborativework, telework and virtual community. Annu. Rev. Sociol. 22:213–38

[79] Thibert B, Bredesen DE, del Rio G. Improved prediction of critical residues for protein function based on network and phylogenetic analyses. BMC Bioinformatics 2005;6:213.

[80] Luke, DA, Harris, JK, 2007. Network analysis in public health: history, methods, and applications. Annu Rev Public Health. 28, 69-93.

[81] James K. Dame, 2007, Evaluation of ecological network analysis: Validation of output, Ecological Modelling; Vol. 210 Issue 3, p327-338, 12p

[82] Fell,D.A. (1992) Metabolic control analysis – a survey of its theoretical and experimental development. Biochem. J., 286, 313-330

[83] Brand,M.: Fast online svd revisions for lightweight recommender systems. In: Proceedings of the 3rd SIAM International Conference on Data Mining. (2003)

[84] T. Kavitha, C. Liebchen, K. Mehlhorn, D. rios Michail, R. Rizzi, T. Ueckerdt, and K. A. Zweig, Cycle bases in graphs: Characterization, algorithms, complexity, and applications, Computer Science Review 3 (2009), 199–243.

[85] P. Hingston, Using Finite State Automata for Sequence Mining, In: Proceedings of the 25th Australasian Computer Science Conference, Melbourne, Australia, 105 – 110, 2001.

[86] M. Lange, H. Leiß, To CNF or not to CNF? An Efficient Yet Presentable Version of the CYK Algorithm, informatica didactica 8, 2009, http://www.informatica-didactica.de

[87] David Luper, Caner Kazanci, John Schramski, Hamid R. Arabnia, Flow Decomposition in Complex Systems, ITNG 2011

[88] Schramski J.R., Gattie, D.K., Patten, B.C., Borret, S.R., Fath, B.D., (2006). Indirect effects and distributed control in ecosystems: distributed control in the environ networks of a seven-compartment model of nitrogen flow in the Neuse River Estuary. USA-steady-state analysis. Ecol. Medel. 194, 189-201.

[89] Kazanci, C., EcoNet: A new software for ecological modeling, simulation and network analysis, Ecol. Model., vol. 208, no. 1, pp. 3--8, 2007.

[90] John Cocke, Jacob T. Schwartz (1970). Programming languages and their compilers: Preliminary notes. Technical report, Courant Institute of Mathematical Sciences, New York University.

[91] Tadao Kasami (1965). An efficient recognition and syntax-analysis algorithm for context-free languages. Scientific report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA.

[92] Danial H. Younger (1976). Recognition and parsing of context-free languages in time n3. Information and Control 10(2): 189-208.

[93] Tollner, E. W. and Schramski, J. R. and Kazanci, C. and Patten, B. C., Implications of network particle tracking (NPT) for ecological model interpretation, Ecological Modelling, vol. 220, no. 16, pp. 1904--1912, 2009.

[94] L. B. Booker, D. E. Goldberg, and J. H. Holland, “Classifier systems and genetic algorithms,” in Machine Learning: Paradigms and Methods, J. G. Carbonell, Ed. Cambridge, MA: MIT Press/Elsevier, 1989, pp. 235–282.

[95] Guetlein, M., Frank, E., Hall, M. Karwath, A.: Large Scale Attribute Selection Using Wrappers. Proc IEEE Symposium on Computational Intelligence and Data Mining, 332-339 (2009)

[96] M.A. Hall and G. Holmes, “Benchmarking Attribute Selection Techniques for Discrete Class Data Mining,” IEEE Trans. Knowledge and Data Eng., vol. 15, no. 6, pp. 1437-1447, Nov./Dec. 2003.

[97] García-López, F., García-Torres, M., Moreno-Pérez, J.A. & Moreno-Vega, J.M. (2004) Scatter Search for the feature selection problem. Lecture Notes in Artificial Intelligence, 3040, 517-525

[98] GUYON, I. AND ELISSEEFF, A. 2003. An introduction to variable and feature selection. J. Mach. Learn. Res. 3, 1157–1182.

[99] Kazanci, C. and Matamba, L. and Tollner, E. W., Cycling in ecosystems: An individual based approach, Ecol. Model., vol. 220, no. 21, pp. 2908--2914, 2009.

[100] Plant C, Teipel SJ, Oswald A, Bohm C, Meindl T, Mourao-Miranda J, Bokde AW, Hampel H, Ewers M. Automated detection of brain atrophy patterns based on MRI for the prediction of Alzheimer's disease. Neuroimage. 2010;50:162–174.

[101] Christian, R. R. and Thomas, C. R., 2000. Neuse river estuary modeling and monitoring project stage 1: network analysis for evaluating the consequences of nitrogen loading No. 325-F, Water Resources Research Institute, Raleigh, NC.

[102] Christian, R. R. and Thomas, C. R., 2003. Network analysis of nitrogen inputs and cycling in the Neuse River estuary, North Carolina, USA. Estuaries, 26:815-826.

[103] Reckhow, K. H. and Gray, J., 2000. Neuse River Estuary Modeling and Monitoring Project Stage 1: Stage 1 Executive Summary and Long-term Modeling Recommendations. Report No. 325-A, Water Resources Research Institute of the University of North Carolina, Raleigh, NC.

[104] Ulanowicz, R.E., 1983. Identifying the structure of cycling in ecosystems.Math. Biosci. 65, 219–237.

[105] David Luper, Caner Kazanci, John Schramski, Hamid R. Arabnia, System Decomposition for Temporal Concept Analysis, 19th International Conference on Conceptual Structures, ICCS 2011

[106] Priss, U.: Formal concept analysis in information science. Annual Review of Information Science and Technology, 40, 521–543 (2006).

[107] Neouchi, R., Tawfik, A.Y., Frost, R.A.: Towards a Temporal Extension of Formal Concept Analysis. In Proceedings of the 14th Canadian Conference on Artificial Intelligence, Ottawa, Ontario. (2001)

[108] Wolff, K.E.: Interpretation of Automata in Temporal Concept Analysis. In: U. Priss, D. Corbett, G. Angelova (eds.): Integration and Interfaces. Tenth International Conference on Conceptual Structures. Lecture Notes in Artificial Intelligence. Springer-Verlag 2002.

[109] Wolff, K.E.: Temporal Concept Analysis. In: E. Mephu Nguifo & al. (eds.): ICCS-2001 International Workshop on Concept Lattices-Based Theory, Methods and Tools for Knowledge Discovery in Databases, Stanford University, Palo Alto (CA), 91-107 (2001)

[110] David Luper, Muthukumaran Chandrasekaran, Khaled Rasheed, and Hamid R. Arabnia, Path Normalcy Analysis Using Nearest Neighbor Outlier Detection, ICAI 2008, 2008, p. 776-783, Proc. of International Conference on Information and Knowledge Engineering (IKE'08; ISBN #: 1-60132-075-2), Las Vegas, USA, July 14-17, 2008 pp. 776-783.

[111] David Luper, Ron McClendon, and Hamid R. Arabnia, Positional Forecasting From Logged Training Data Using Probabilistic Neural Networks, Proc. of International

Conference on Information and Knowledge Engineering (IKE'09; ISBN # for set: 1-60132-116-3), Las Vegas, USA, July13-26, 2009, pp. 179-189.

[112] R. P. Rohr, H. Scherer, P. Kehrli, C. Mazza and L. Bersie, Modeling Food Webs: Exploring Unexplained Structure Using Latent Traits, The American Naturalist, Vol. 176, No. 2, August 2010, (pp. 170-177)

[113] David Temperley, Modeling Common - Practice Rhythm, Music Perception: An Interdisciplinary Journal, Vol. 27, No.5, June 2010, p. 355 – 376

[114] Marcelo C. Medeiros and Alvaro Veiga, Modeling Multiple Regimes in Financial Volatility with a Flexible Coefficient GARCH(1, 1) Model, Econometric Theory, Vol. 25, No. 1, Feb. 2009, p 117 – 161

[115] Ashbrook, D., Starner, T.: Using GPS to learn significant locations and predict movement across multiple users. Personal and Ubiquitous Computing 7 (2003) 275–286

[116] Whipple, S.J., Patten, B.C. , Borrett, S.R. Unpublished manuscript. Indirect effects and distributed control in ecosystems- Comparative network environ analysis of a seven-compartment model of nitrogen storage in the Neuse River estuary, USA: Time series analysis.

[117] Christian, R. R. and Thomas, C. R., 2003. Network analysis of nitrogen inputs and cycling in the Neuse River estuary, North Carolina, USA. Estuaries, 26:815-826.

[118] Corbett, D.R., 2010. Resuspension and estuarine nutrient cycling: insights from the Neuse River Estuary. Biogeosciences, 7:3289-3300.

[119] Kemp, W.M., Sampou, P., Caffrey, J., and Mayer, M. 1990. Ammonium recycling versus denitrification in Chesapeake Bay sediments. Limnology and Oceangraphy 35:1545-1563.

[120] Luettich Jr., R. A., McNinch, J. E., Paerl, H., Peterson, C. H., Wells, J. T., Alperin, M., Martens, C. S. and Pinckney, J. L., 2000. Neuse River Estuary modeling and monitoring project stage 1: Hydrography and circulation, water column nutrients and productivity, sedimentary processes and benthic-pelagic coupling, and benthic ecology No. 325-B, Water Resources Research Institute of the University of North Carolina, Raleigh, NC.

[121] Rizzo, W. M. and Christian, R. R., 1996. Significance of subtidal sediments to heterotrophically-mediated oxygen and nutrient dynamics in a temperate estuary. Estuaries, 19:475-487.

[122] Baird, D., McGlade, J .M. a n d Ulanowicz, R.E., 1991. The comparative ecology of six marine ecosystems. Phil. Trans. R. Soc. Lond., 333: 15-29.

[123] Kolding, J., 1993. Trophic interrelationships and community structure at two different periods of Lake Turkana, Kenya - a comparison using the ECOPATH II box model. In: V. Christensen and D. Pauly (Editors), Trophic Models of Aquatic Ecosystems. ICLARM Conf. Proc. 26, pp. 116 - 123.

[124] Bayley, P.B. 1977. Changes in species composition of the yields and catch per unit effort during the development of the fishery at Lake Turkana, Kenya. Arch. Hydrobiol, 79:111-132.

[125] Hopson, A.J., editor. 1982 Lake Turkana – a report on the findings of the Lake Turkana Project 1972-1975, Vols. 1-6. Overseas Development Administration, London. 1614 p.

[126] Kallqvist, T., L. Lien and D. Litti 1988. Lacke Turkana – limnological 1985-1988. Norwegian Institute for Water Research (NIVA), Kenya Marine and Fisheries Research Institute. NIVA Rep. No. 0-85313, 97 p. Oslo, Norway

[127] Kolding, J. 1989. The fish resources of Lake Turkana and their environment. Cand. Sci. Thesis and final report of KEN 043 Trial Fishery 1986-1987. 262 p. University of Bergen, Norway.

[128] Boling R.H, Goodman E.D, Van Sickle J.A, Zimmer J.O, Cummings K.W, Petersen R.C, and Reice S.R. 1975. Toward a model of detritus processing in a woodland stream. Ecology. 56(1):141-151.

[129] Steinberg D.K, Silver M.W, Palskaln C.H, Coale S.L, and Paduan J.B. 1994. Midwater zooplankton communities on pelagic detritus in Monterey Bay, California. Limnology and Oceanography. 39(7)1606-1620.

[130] Allesina S, Bondavalli C, Scharler U.M, 2005. The consequences of the aggregation of detritus pools in ecological networks. Ecological Modelling. 189:221-232.

[131] Dunne, J.A., Williams, R.J., Martinez, N.D., 2002. Food-web structure and network theory: the role of connectance and size. Proc. Natl. Acad. Sci. U.S.A. 99, 12917–12922.

[132] Dunne, J.A., Williams, R.J., Martinez, N.D., 2004. Network structure and robustness of marine food webs. Mar. Ecol. Prog. Ser. 273, 291–302

[133] Fath, B.D., Halnes, G., 2007. Cyclic energy pathways in ecological food webs. Ecol. Model. 208, 17–24.

[134] Halnes G, Fath B.D, Liljenström H. 2007. The modified niche model: Including detritus in simple structural food web models. Ecological Modelling. 208:9-16.

[135] N. Packard and A. Perelson, (1986) "The immune system, adaptation and machine learning", Physica D, vol. 2, pp. 187–204

APPENDIX

The appendix will serve to provide tutorials on how to use the code developed for the methodologies proposed in this work. The code is executed from a terminal prompt and depends on command flags being passed into the program. The command line flags will not work if any directories used by the program (run time, output, *etc*.) have spaces in them; so don't use spaces in any directory structures used by this program.

A: Functional Decomposition Tutorial

This section is a tutorial for using the code developed to perform the structural and functional decompositions outlined in this work. The decompositions are the first steps in the Network Flux Analysis methodology outlined in the application chapter. After reading this tutorial, a user should be able to download the code and calculate a functional decomposition vector from a distribution of symbolic network pathways (a distribution of symbolic time series data can also be used).

To use the code go to the following URL and download the project: https://github.com/davidluper/system-decomposition-python (the code is hosted as a repository on github). The code is written in Python (v. 26) so this will need to be installed on the system being used to run the code. Additionally, this code uses the NumPy scientific computing package which also needs to be installed on the system being used.

Once the project is downloaded and the Python environment is set up on the machine, it is possible to use the code to functionally decompose a system. To run the code, open a terminal window on the system and navigate to a directory where a Python script can be executed. If the environment variables are configured on the system, then it may be possible to run Python scripts

from any directory. If the environment variables are not configured, then the user may need to be in the Python directory in order to execute a Python script.

After verifying the ability to run a Python script, the user can invoke the functional decomposition routines using the following command line:

- python [path to decomp script]\system_decomp.py -decompavg -dir [directory to use for the decomp] -paths [name of file containing the pathway data] -map [comma separated symbol ordering string]
- Example of this command
    - C:\Python26>python k:\decomp_release\system_decomp.py
      -decompavg
      -dir k:\decomp_release\out\
      -paths oyster-reef_completed_paths.dat
      -map 0,1,2,3,4,5

The command line above will execute a functional decomposition using the averaging technique outlined in this work to deal with any grammar ambiguities that arise. In the github repository, there are example pathway data files that can be used for tutorial purposes. The -dir command flag specifies a directory that will house the pathway data file and any output generated by the functional decomposition code. The -paths command flag specifies the name of the file housing the pathway data (in the case of this example it is a pathway file output from an ecological system simulated with Network Particle Tracking). This file needs to be formatted such that each pathway is terminated with a carriage return/new line and the symbols in the pathway are delimited with a space. To make the output more readable to humans, the -map command flag specifies a logical ordering of symbols to use in the methodology. For the

example oyster-reef pathway file, the symbols are numbers from 0 to 5. These have an obvious numeric ordering that can be specified to the decomposition routines. The format for this flag is to list the symbols from left to right in their desired order delimiting them with a comma. An example of this for the oyster-reef sample data would be -map 0,1,2,3,4,5.

Once executed, the decomposition will read and parse all pathways in the distribution contained in the pathway data file specified. This can be quite lengthy depending on the complexity of the system being decomposed. Output to the screen will indicate the number of pathways parsed as well as a time stamp. This output will occur on an interval of 100 pathway parses. The calculated coefficients vector will be output in CSV format in a file called out.csv. The header line in this file will indicate the actual fluxes to which the coefficients apply.

Once coefficients are computed they can be used for data mining and knowledge extraction. The NFA methodology proposed in this work utilized WEKA for supervised learning over a collection of coefficient vectors. The coefficient vectors were labeled seasonally and feature extraction was performed. The features selected to optimally classify the vectors are seen as encoding the values (summer, spring, fall, winter) of that high level system trait (seasonality). A Support Vector Machine was used both in feature selection and to produce the final classification result verifying how accurately a subset of features performed at classifying the data.

A side note that needs to be mentioned is that the pathway data should not have the environment listed in the pathways. This was due to a convention adopted from Network Particle Tracking. A path through the system such as Environment > 0 > 2 > 1 > Environment should be listed in the pathway file in the following manner:

- 0 2 1

B: Flux Shift Analysis Tutorial

Flux Shift Analysis (FSA) was presented in this work as a methodology to compare snapshots of a system to see, on a flux by flux basis, how system activity has moved. This tutorial is meant to communicate how to use the code developed for FSA. After reading this tutorial, the user should be able to compute a Constant Distance Vector (CDV) for a system that has had two functional decompositions performed on it.

To use the code go to the following URL and download the project: https://github.com/davidluper/system-decomposition-python (the code is hosted as a repository on github). The code is written in Python (v. 26) so this will need to be installed on the system being used to run the code. Additionally, this code uses the NumPy scientific computing package which also needs to be installed on the system being used.

Computing the Difference Vector Matrix

The ability to compute a CDV relies on a construct called the Difference Vector Matrix (DVM). The DVM can be calculated by the code in two different ways. The first way is to run an exhaustive search. The runtime for an exhaustive search is factorial on the number of internal cycles that exist within the system (internal cycles are cycles not containing the environment). The application chapter of this work applied an exhaustive search to the Lake Turkana ecosystem, and that system push the bounds of computability for the exhaustive search. That system had 125 total fluxes and 19 of them were internal cycles. The methodology took around 24 hours to compute the DVM using this approach. To compute the DVM with an exhaustive search use the following command line:

- python [path to decomp script]\system_decomp.py -dvm -dir [directory to use for the decomp] -paths [name of file containing the pathway data] -map [comma separated symbol ordering string]
- Example of this command
    - ```
      C:\Python26>python k:\decomp_release\system_decomp.py -dvm
      -dir k:\decomp_release\dvm\
      -paths oyster-reef_completed_paths.dat
      -map 0,1,2,3,4,5
      ```

For clarification on the -dir, -paths, and -map command line flags, see the functional decomposition tutorial. Running this command will create a file in the output directory called dv-log.txt that contains the exhaustively computed DVM.

The second approach to computing the DVM relies on a standard decomposition as outlined in the functional decomposition tutorial. This approach computes the DVM inline as it is parsing pathways from the distribution supplied to the decomposition routine. This approach has the same runtime as the functional decomposition, which is far better than an exhaustive search; however, the disadvantage to computing the DVM in this manner is that there is no guarantee all the Difference Vectors will be found. This approach will only find Difference Vectors that occur in the pathway distribution presented to the functional decomposition routine. Difference Vectors happen when a certain combination of fluxes appears within a pathway. Fluxes have a probability of occurring based on the flow weights present in a system's flow vector. It then follows that Difference Vectors also have a probability of occurring. This means that if enough pathways are interpreted, statistically probable Difference Vectors will be found. It is an open research problem as to whether finding statistically probable Difference Vectors is
      ```

good enough, or if an exhaustive search is required. To compute the DVM using an inline approach, execute the following command at the command line:

- python [path to decomp script]\system_decomp.py -decompinlinedvm -dir [directory to use for the decomp] -paths [name of file containing the pathway data] -map [comma separated symbol ordering string]
- Example of this command
    - C:\Python26>python k:\decomp_release\system_decomp.py -decompinlinedvm -dir k:\decomp_release\out\ -paths oyster-reef_completed_paths.dat -map 0,1,2,3,4,5

For clarification on the -dir, -paths, and -map command line flags, see the functional decomposition tutorial. Running this command will create a file in the output directory called dvm-inlinesearch.txt that contains the DVM, which was computed inline during the functional decomposition.

Once the DVM is calculated, the CDV can be computed the following command line:

- python [path to decomp script]\fsa.py [directory for the first decomposition of the system] [directory for the second decomposition of the system]
- An example of this command as applied to a DVM computed with an exhaustive search
    - C:\Python26>python k:\decomp_release\fsa.py k:\decomp_release\fsa1 k:\decomp_release\fsa2
- An example of this command as applied to a DVM computed inline with a functional decomposition (notice the -inline flag)

- ```
  C:\Python26>python k:\decomp_release\fsa.py
  k:\decomp_release\fsa1 k:\decomp_release\fsa2
  -inline
  ```

If the DVM being used was computed inline, then -inline should be appended to the above command line as an additional flag (as seen in the second example command). Also, the DVM file (dv-log.txt or dvm-inlinesearch.txt) should be in the first directory specified or it will not be found. The output for this command will produce two files, distances_absolute.csv and distances_relative.csv. The absolute distances file uses the raw frequency count at each flux to compute the CDV. The relative distances file uses normalized frequency counts to compute the CDV where a coefficient is equal to itself divided by the sum of all the coefficients in the vector to which it belongs.

C: Reverse Transform Tutorial

Reversing a functional decomposition is mentioned in the application and conclusion sections of this work. This tutorial is meant to inform a user as to how to obtain a modified adjacency matrix built from the reverse transform of a functional decomposition.

To use the code go to the following URL and download the project: https://github.com/davidluper/system-decomposition-python (the code is hosted as a repository on github). The code is written in Python (v. 26) so this will need to be installed on the system being used to run the code. Additionally, this code uses the NumPy scientific computing package which also needs to be installed on the system being used.

In order to perform a reverse transform of a functional decomposition, a user must first perform a functional decomposition. For information on how to accomplish this, see the functional decomposition tutorial. Once a functional decomposition has been performed, it is
  ```

possible to reverse the functional decomposition by mapping selected coefficients back into an adjacency matrix, while leaving unselected coefficients out. Which coefficients are mapped back into the modified adjacency matrix are specified by the user through a command flag. To compute a reverse transform use the following command:

- python [path to decomp script]\system_decomp.py -revtran [directory housing the functional decomposition output] [comma separated list of coefficient indexes to map back into the modified adjacency matrix]
- Example of this command
    - `C:\Python26>python k:\decomp_release\system_decomp.py -revtran k:\decomp_release\revtran\ 0,2`

Running this command will produce an output file called rev_matrix.txt that houses the modified adjacency matrix. Uses for the modified adjacency matrix need to be researched, but one use, in relation to NFA, is to analyze the percentage of flow kept along particular edges after mapping back isolated coefficients encoding high level system traits. Isolating fluxes and reversing the transform in this way could help understand, for each edge in a system, the percentage of flow being used at that edge towards understanding a high level system trait.