Evolving Software Traders and Detecting Community Structure in Financial Markets

by

Todd D. Kaplan

A.B., Biology, Brown University, 1993
Diploma, Computer Science, Cambridge University (UK), 1996
M.Sc., Applied Mathematics, Oxford University (UK), 2002

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

May, 2011

Dedication

I dedicate this work to my grandparents – Henrietta and Ralph Kaplan along with Morris and Edith Lotner – all of whom passed during this endeavor. If not for their love and sacrifices, this journey would not have been possible. I also memorialize my friend Jonathan Collins Hoare, whose grace, friendship, and spirit are not forgotten by those he touched.

Acknowledgments

Firstly, I would like to thank my committee. My chair, Stephanie Forrest, kept her faith in me despite trying moments. She provided me the academic freedom to explore and find my own path – and, in the end, she helped me reach the finish line. For both, I am grateful. Before starting my Ph.D., I had read a book about Doyne Farmer – it insprired me to undertake this journey. It was a privilege to work with him and he helped me improve myself from scientific tourist to practitioner. Cristopher Moore and Terran Lane both provided needed encouragement at key times.

The Adaptive Computation Group at the University of New Mexico, led by Stephanie Forrest, has been my home for quite some time. A lot of the names and faces have changed and there are many to thank. I greatly credit our weekly seminar meetings for helping me develop both as a scientist and a colleague.

I would not have reached this goal without the friendship and mentorship of my good friend Derek J. Smith. From Albuquerque to Las Vegas to California to England, he made science "fun". I had a Neuroscience professor at Brown University that deemed himself a "neuro cowboy". In that light, I nominate Derek for the title of "computation cowboy". Derek taught me that no question is too simple. Oh, and D, they call it a "court", not a "pitch". And, each team has five players on the court at all times.

Fernando "Don Futbol" Esponda deserves many thanks. His friendship and scientific comradery helped me persevere. However, I must qualify this acknowledgment: Nando's theory that tequila cures the common cold is a farce. Certainly, do not apply the methodology the night before delivering a talk.

Finally, I would like to thank the University of New Mexico Center for High Performance Computing for providing computational cycles to conduct the experiments.

Evolving Software Traders and Detecting Community Structure in Financial Markets

by

Todd D. Kaplan

ABSTRACT OF DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

May, 2011

Evolving Software Traders and Detecting Community Structure in Financial Markets

by

Todd D. Kaplan

A.B., Biology, Brown University, 1993

Diploma, Computer Science, Cambridge University (UK), 1996
M.Sc., Applied Mathematics, Oxford University (UK), 2002

Ph.D., Computer Science, University of New Mexico, 2011

Abstract

A trophic network, commonly referred to as a food web, describes the feeding relationships between different groups of species in an ecosystem. Ecologists construct trophic networks to aid their understanding of ecosystems. In the realm of financial markets, trophic networks can serve an analogous role. Their use could potentially illuminate underlying dynamics responsible for commonly observed macro-level phenomena. For example, one might hypothesize that periods of market volatility occur after a keystone trader species becomes inactive and the trophic network subsequently restructures itself. The primary topic in this research investigates whether it is possible to detect trophic structure within real-world financial markets. Secondarily, the efficacy of using genetic programming to evolve software traders in a simulated stock market (continuous double auction) is examined.

The research to follow is split into three parts. In Part I, new tools for detecting community structure in complex networks are developed. First, a two-phase macro-strategy for community detection is introduced. The approach is unique in that it can be used in combination with any existing community detection algorithm to provide high-yield, robust results. Second,the resolution limit inherent to the community structure measurement known as modularity is illustrated experimentally. To overcome this limitation, a fine-granularity community structure measure called divisionality is developed. Third, a dual-assortative measure (DAMM) of community structure is established. DAMM extends the domain of networks that can be analyzed for community structure to include those with negatively weighted edges.

Part II focuses on the evolution of software agents that compete in an artificial financial market. The evolutionary framework is based on a stack-based language (Staq) that was developed for genetic programming (GP). The genetic programs of two evolved agents, each based on a different fitness function, are examined. One of these evolved traders, known as clear & hoist (CH), reveals a limitation of the simulated market: a lack of fundamentalism. Two value-based strategies are developed to address this shortcoming. The effect of each strategy on the CH trader is independently examined.

In Part III, the community structure tools developed in Part I are used to detect trophic species in financial market data. After introducing the trophic detection algorithm, a methodology for assessing the significance of detected structure is described. The efficacy of the approach is demonstrated using simulated data. Finally, real-world data from the London Stock Exchange (LSE) is examined using the trophic detection framework. Although significant structure is detected in subsets of the real-world data, the results are inconsistent. However, given limitations of the LSE data, the lack of consistent detection is not surprising. Most notably, each trader in this data represents an entity acting on the behalf of many individuals and institutions having different strategies. Due to this aggregation, the trading actions of individuals are obfuscated and thus the trophic structure is as well. Examination of real-world data with greater specificity – detailing trades at the level of individuals – is warranted.

Contents

List of Figures

List of Tables

Glossary

agglomerative algorithm	A community structure algorithmic approach whereby each node is initially declared to comprise an independent community. Thereafter, on each step of the algorithm, two communities are joined. The algorithm halts once it is no longer possible to improve the prescribed community structure measure. When viewing a dendrogram, an agglomerative algorithm follows a bottom-up approach.
ask price	The highest buy price at any time t. Denoted as $a(t)$.
assortativity	The tendency for nodes to be attached to other nodes that are like (or unlike) them in some way.
bid price	The lowest selling price at any time t. Denoted as $b(t)$.
continuous double auction (CDA)	The continuous double auction is the most widely used market clearing mechanism in modern financial markets. The auction is considered *double* because traders can submit both buy and sell orders, and it is considered *continuous* because traders may submit orders at any time. Orders supplied to the CDA are cleared using a limit order book (LOB). Figure 6.1 illustrates a LOB.

clustered volatility	Refers to large market price swings grouped within a short time period.
CNM	An agglomerative algorithm introduced by Clauset, Moore, and Newman [11].
communal overlap	A measure of similarity between known and detected communities that I develop in section 5.2.3. Denoted as Ω.
deterministic division (DD)	Deterministic division is a divisive community detection algorithm that I establish in section 3.
degree	The number of edges connected to a node. In a directed network, degree is sub-classified into in-degree and out-degree.
dendrogram	A tree-like representation of the hierarchical organization of data; in the context of networks, the vertices of the network are arranged as the leaves of the tree, with the branches of the tree denoting their hierarchical clustering. Can be drawn in either Cartesian or polar coordinates.
divisionality	A fine-granularity measure of community structure that I developed. Divisionality is introduced in Chapter 4. It is denoted as D.
divisive algorithm	A community structure algorithmic approach whereby all nodes are initially placed into a single community. Thereafter, each community is recursively divided until an improvement in the prescribed community structure measure can no longer be improved. At that point, a community is declared indivisible. Once all communities have been declared indivisible, the algorithm halts. When viewing a dendrogram, a divisive algorithm follows a top-down approach.

extremal optimization (EO)	Extremal optimization is a stochastic algorithm which is used in this dissertation to optimize measures of community structure (e.g. modularity or divisionality).
fundamentalist	A trader that trades according to discrepancies in the perceived value of a stock and the market price. If the trader believes a stock is undervalued, it is inclined to buy in anticipation that the price will rise. If the trader believes a stock is overvalued, the trader is inclined to sell. A fundamentalist is sometimes referred to as a value trader.
limit order book (LOB)	The data structure used in a CDA to cache limit orders. Orders are prioritized first by price and then by time.
limit order	An that order specifies the following fields: the quantity of shares to be traded, whether to buy or sell, a limit price, and an expiration time. Limit orders do not generate immediate transactions. Rather, they are cached on a limit order book according to price and time of arrival.
liquidity	The ability of a market to accept large transaction with minimal to no impact on price stability.
LONT	A noise trader that places limit orders only.
LSE	London Stock Exchange.
market order	Market orders specify only two fields: the quantity of shares to be traded and whether to buy or sell. Significantly, market orders do not designate a transaction price. Market trigger immediate transactions. The transaction price is dictated by the best available price on the opposing side of the limit order book.

midpoint price	The midpoint of the bid and ask prices, $p(t) = \frac{a(t)+b(t)}{2}$. It is commonly used as an approximation of the market price.
modularity	A statistical measure of community structure proposed by Newman and Girvan that is widely adopted in the complex network community. It is denoted as Q.
MONT	A noise trader that places market orders only.
mutual information	An information theoretic concept that measures the mutual dependence of two random variables.
market maker	A trader that simultaneously buys and sells with the intent of securing a roundtrip profit. The market maker enacts the proverb "buy low, sell high".
noise trader	A trader that places market orders according to a stochastic process. Also referred to as a zero-intelligence trader. Introduced in Chapter 6.
spread	The gap between the bid and ask prices, denoted as $s(t) = a(t) - b(t)$.
Staq	A stack-based language that I designed for genetic programming. Staq provides functionality for both extensible data types and dynamic function creation.
symmetric uncertainty	A normalized variant of mutual information.
trend trader	A trader that places market orders based on patterns observed in market data. In the context of this work, trend traders place orders based on patterns of past price movements. Sometimes referred to as technical traders. Introduced in chapter 8.

trophic species	Ecologically, *trophic species* are functional groups containing organisms that both consume and are consumed by identical species within a food web. Trophic species differ from taxonomic species, which are grouped according to morphology.
trophic web	A network used to describe the feeding relationships between different species within an ecosystem. Commonly referred to as a *food web*.
value trader	See fundamentalist.
VLONT	A limit order placing noise trader that maintains a perceived value of the asset being traded. The trader uses this value to establish the prices at which it places orders.
zero-intelligence trader	See noise trader.

Chapter 1

Introduction

A stock market resembles an ecosystem [15]. An ecosystem is defined by the interaction of its organisms and the environment. In financial markets, the organisms are traders and the environment is defined by the state of the market through which traders interact. Feedback loops exist between organisms and their environment. Traders affect their environment through trades. Changes in the market environment affect future decisions of traders. Ecosystems, as well as financial markets, display many of the traits associated with a *complex adaptive system* [32]: aggregation, adaptation, feedback loops, and nonlinearity.

A trophic network, commonly referred to as a food web, describes the feeding relationships between different groups of species in an ecosystem. Ecologists construct trophic networks to aid their understanding of ecosystems. In the realm of financial markets, trophic networks can serve an analogous role. Their use could potentially illuminate underlying dynamics responsible for commonly observed macro-level phenomena[1]. For example, one might hypothesize that periods of clustered volatility[2] occur after a keystone trader species[3] becomes inactive and the trophic network subsequently restructures itself. The main objective of my research investigates the following question: is it possible to detect trophic structure within a real-world financial market? Secondarily, I examine the efficacy of using genetic programming to evolve software traders in a simulated continuous double auction[4].

The material presented in this dissertation is split into three parts. Part I introduces new algorithms and statistical measures for detecting community structure in complex networks. Part II focuses on the evolution of software agents that compete in an artificial financial market. Part III represents a confluence of the first two parts; I use the tools developed in Part I to analyze financial market data with the objective of unveiling trophic structure. Below, I introduce the content chapters to follow. Chapter 2 provides a literature review; Chapter 9 summarizes and concludes.

As noted, Part I focuses on tools for community detection in *complex networks*. Networks are comprised of nodes and edges. Nodes represent independent entities. Edges, which may or may not be weighted, represent interactions between nodes. For example, consider a friendship network. Here, people are represented with nodes. Friendships between individuals are represented by edges. The term “complex network” refers to a network that has non-trivial topological properties. A lattice, which has roughly the same degree (number of connected edges) at each node, is not consid-

[1]Basic *stylized facts* [22] commonly ascribed to financial markets include: fat-tailed probability distributions of price returns, fast decay of autocorrelation of price returns, slow decay of autocorrelation of absolute value of price returns, slow decay of autocorrelation of square of price returns, and clustered volatility.

[2]Clustered volatility refers to the phenomena of large price swings occurring in a short time span.

[3]A species critical to the operation of an ecosystem. In a biological setting, one example is that of the field vole, the main prey of very wide predators of woodland edges and farmland, such as barn owls.

[4]See glossary for explanation.

ered a complex network because it has trivial topological structure. Characteristics commonly ascribed to complex networks include heavy-tail degree-distribution [37], high clustering coefficient [37], small geodesic distance [37], assortativity of node connections [36] [35], and the presence of community structure [40].

Community structure within a complex network refers to the existence of groups of nodes that are highly connected – more highly connected than had edges of the network been randomly distributed. For example, in a friendship network, a group of people demonstrating a greater than expected number of friendships could be considered a community. The task of *community detection* involves two important components. First, a mathematical definition of community must be determined. Without such a measure, it would be impossible to quantify the success of detection. Second, an algorithm for the detection process is required. In the three chapters of Part I, I address limitations in the existing tools for community detection and propose improvements.

Chapter 3 presents a two-phase strategy for the detection of communities in complex networks. In the first phase, any of the existing algorithms for community detection are applied. In the second phase, the initial result is optimized by inducing fractures (for divisive approaches), or merges (for agglomerative approaches), in the detected communities and then optimizing the resulting constituents. I demonstrate that the two-phase approach achieves consistent, high yield results. Using the second phase alone, it is possible to incrementally improve upon an existing community detection result. As a final contribution, I introduce a divisive algorithm, called *deterministic division*, that provides fast, high yield results when used in combination with the macro-strategy.

In Chapter 4, I demonstrate experimentally that modularity suffers from a *resolution limit*. Because of this limit, optimization of modularity only detects coarse-granularity community structure. I introduce a new measure of community structure, called *divisionality*, that allows for the detection of fine-granularity community structure. The efficacy of the two measurements is demonstrated and compared on graphs with known structure.

Current community detection algorithms operate by optimizing a statistic called *modularity*, which analyzes the distribution of positively weighted edges in a network. Modularity does not account for negatively weighted edges. In Chapter 5, I introduce a *dual assortative modularity measure (DAMM)* that incorporates both positively and negatively weighted edges. After defining and explaining the DAMM statistic, its utility is illustrated using a community detection algorithm. I evaluate the efficacy of the algorithm on both computer generated and real-world networks, showing that DAMM broadens the domain of networks that can be analyzed by community detection algorithms. Additionally, I introduce a statistic called *communal overlap*, which measures the similarity between known and detected sets of communities.

In Part II, I shift focus from complex networks to financial markets. These two chapters (6 and 7), provide stepping stones towards coevolving a population of traders. In Chapter 6, I describe a stack-based genetic programming implementation for evolving market trading strategies. A generational genetic programming approach is used to evolve a single trader that can regularly profit in a continuous double auction market environment inhabited by noise traders. Two fitness functions are investigated. The first function is based on the economic utility measure known as Sharpe ratio. The second function incorporates a penalty for lack of inventory control. The genetic programs of two exemplar agents, each relating to a different fitness function, are described. The market performance of each agent is examined. Despite their vastly different strategic approaches, both agents achieve consistent profits at the expense of the noise traders.

Chapter 7 addresses a limitation inherent in the simulated market environment.

Without the inventory control penalty, a consistent GP solution evolves. This evolutionary strategy, that I call clear & hoist (CH), succeeds due to a lack of fundamentalism[5] in the simulated market. To remedy this shortcoming, I introduce two value-based trading strategies for the continuous double auction (CDA). The mechanics of both strategies are described and each is independently examined in a market environment inhabited by noise traders. After establishing that both approaches successfully encourage the market price to track a perceived value[6], the effect that each strategy has on the CH trader is examined. The merits of the two value-based approaches are compared.

In Part III, I apply the tools developed in Part I to financial market data with the goal of detecting trophic structure. First, methods used for detecting trophic species in financial market data are introduced. Then, I discuss a methodology for assessing the significance of the detected structure. The efficacy of the methods is examined using two types of data. First, I demonstrate that the methods successfully detect different trader types in data generated by the simulator used in Chapters 6 and 7. In this study, all of the software traders are hard-coded – no evolutionary agents are included. Next, I examine data from the Shell (SHEL) stock of the London Stock Exchange (LSE). I demonstrate that it is indeed possible to detect significant trophic structure within month-long segments of the real-world data. However, significant structure is not detected in all of the examined periods. The lack of consistent detection is not surprising given limitations of the LSE data. Most notably, each trader in this data represents an entity acting on the behalf of many individuals and institutions having different strategies. Because this aggregation, the trading patterns of individual traders are obfuscated and thus the trophic structure is as well. Ideally, data with greater specificity – detailing the trading of individuals –

[5]Fundamentalist traders associate a perceived value with the asset being traded.

[6]Each value-based trader associates a perceived value with the asset being traded. The actions of each value-based strategy encourage market price to converge towards this perceived value.

would be analyzed[7].

[7]Data with this level of specificity was not available for this research.

Chapter 2

Literature Review

This chapter gives an overview of the literature relevant to my dissertation. It is split into three sections, each associated with one of three parts of this dissertation. Where applicable, the reader is referred to material in the main content chapters for more complete descriptions.

2.1 Tools for detecting community structure in complex networks

Newman and Girvan introduced the concept of community structure in complex networks in [40]. *Hierarchal clustering* [59] [49], the traditional method for detecting structure of this kind, fails to correctly identify community structure in certain cases [40]. In particular, hierarchal clustering frequently isolates peripheral nodes and places them in their own community, rather than associating them with communities to which they are attached [40]. Newman and Girvan present an algorithm that iteratively removes edges for detecting communities and suggest an explanation for community structure based on assortative mixing. In [34], Newman and Girvan propose the statistical measure of community structure called *modularity*. Further, they describe three algorithms for the detection of community structure. Each of these algorithms involves removing edges based on a measure called *edge betweenness*. The best algorithm has an asymptotic running time of $O(m^2n)$, where m represents the number of edges and n represents the number of nodes. In that research, the authors use modularity to quantify the success of a community partitioning after the completion of an algorithm. Newman [33] shifts the algorithmic approach by introducing an agglomerative approach that optimizes modularity. The algorithm improves asymptotic running time to $O((m+n)n)$. Brandes et al [8] show that maximizing modularity is NP-complete. Clauset et al [11] describe a hierarchal agglomerative algorithm (CNM) that runs in $O(md \cdot \log(n))$, where d represents the depth of the resulting dendrogram[1]. Duch and Arenas [14] introduce a divisive approach based on extremal optimization [6] [7] that runs more slowly than CNM but provides markedly better results. In [39] [38], Newman introduces spectral-based methods that provide competitive results to that of Duch and Arenas. A Kernighan-Lin [24] optimization is used to refine the result of each spectral-based division. Reichardt and Bornholdt [45] propose a community detection method based on a q-state Potts model. They show that communities can be evaluated using the Hamiltonian of a modified Potts spin glass model. My macro-strategy of Chapter 3 presents an orthogonal approach that can be combined with any existing algorithms, whether it be agglomerative or divisive.

With regards to statistical measures of community structure, Fortunato [18] illustrates the *resolution limit* inherent to modularity through analysis. Because of this limit, optimization of modularity yields a coarse-grained community partitioning. Kumpula [28] et al show that a resolution limit exists for the methods of Reichardt

[1]A dendrogram is a branching diagram used to represent the hierarchal relationships between communities.

and Bornholdt as well. Arenas et al [2] present a measure that can be used to detect community structure at different scales. A parameter, r, is required to specify the scale at which to detect structure. My divisionality measure of Chapter 4, developed independently, presents a fine-granularity community structure measure.

Newman [35] [36] demonstrates *assortativity*, the tendency for nodes to be attached to other nodes that are like (or unlike) them in some way. I build on this concept to establish the dual-assortative community structure measure (DAMM) presented in Chapter 5. With reference to clustering algorithms, Bansal et al [4] establish a clustering measure that integrates the contributions of both positive and negative weights. In [30], rather than concentrating on edge weights, Leicht and Newman focus on detecting community structure in directed networks.

2.2 Evolving traders in a simulated financial market

2.2.1 Genetic programming (GP)

In Part II, I adopt genetic programming as a framework for evolving software traders. Koza [25] [27] [26] founded the GP using a tree-based approach. The stack-based language that I built for GP (called Staq and introduced in Chapter 6), bears strong resemblance to the *Push* [55] language designed by Spector, which descended from *Forth* [48]. Like Push, Staq provides functionality for extensible types, dynamic function creation, and looping constructs. Construction of Staq began before publication of Push; however, an effort was made to align the Staq instruction set to that of the second version of Push.

2.2.2 Santa Fe Artificial Stock Market (SFASM)

One of the earliest computational models of a stock market is the Santa Fe Artificial Stock Market [3] [41] [42]. The SFASM is based on a centralized market maker clearing mechanism, as opposed to a continuous double auction (CDA). Each trader, or agent, maintains a strategy represented as a ternary string of market indicators. Strategies are evolved using a genetic algorithm. Agents are given the choice to either invest in a single available stock or leave their money in a bank. The stock pays a stochastic dividend and its price fluctuates according to agent demand. The bank pays a fixed interest rate.

The objective of the SFASM experiments was to examine whether or not the agents settle into behavior predicted by general equilibrium theory or rather the more complex behavior found in real financial markets. The answer is that both are possible; however, the rational expectations behavior is a local attractor that requires both restricted initial parameter settings and a low exploration rate by the genetic algorithm. The dominant attractor for market behavior is that of the complex regime of real markets – displaying fat tails in price returns and higher trading volume.

2.2.3 Genetic programming applied to finance

Andrews and Prager [1] were among the first to apply genetic programming in the realm of computational finance. A simplified double auction (DA) environment is employed. In each experiment, a single GP trader is evolved in a market inhabited by hand-coded strategies[2]. For each auction game, four traders are designated as buyers and four others as sellers. The GP agent always assumes the role of a seller. The game is played in a series of rounds. Tokens are the commodity traded. Prices

[2]The strategies were successful entrants from the double auction tournaments held at the Santa Fe Institute in 1990 [46] [47].

are represented in integer units. At the beginning of a game, each trader is assigned values for the tokens that they are to buy or sell. Trading takes place in discrete time steps. At the start of each time step, traders are told the current bid and offer, a history of previous trades, and the identities of the last traders. Then, they return their own bid (or offer) values. The new holders of the current bid and offer are then given the option of executing a trade. GP programs are represented by tree structures. The terminal set consists of seven tokens. Six functions comprise the functional set. Each GP trader independently competes in games with the hand-coded strategies. Fitness is determined by comparing the profits of the GP trader to those of the others. A population of 300 GP traders was employed – the fitness of each GP trader by competing in independent games with the hand-coded strategies. Experiments terminate either when a predetermined number of rounds transpired or once a GP strategy matching or exceeding the performance of the hand-coded strategies is identified. Andrews and Prager found that in 65% of their experiments, the GP traders achieved performance exceeding or matching that of the hand-coded traders.

The work of Phelps et al [43] is relevant to coevolving a population of traders. The authors explore the simultaneous coevolution of auction pricing mechanisms and trading strategies using genetic programming. The objective was to discover auction pricing models that are robust to a wide range of trading behaviors. GP programs are represented by tree structures. For trading strategies, the terminal set contained 5 tokens and the functional set contained 10 functions. For auction pricing rules, the terminal set contained 4 tokens and the functional set had a size of six. The fitness of traders is determined by profits earned. The fitness of auction pricing rules is determined by the ratio of profits earned to that theoretically available in competitive equilibrium. The authors demonstrate that market efficiency is achieved faster by the GP auction pricing mechanism in an environment inhabited by GP traders as opposed to an environment inhabited by fixed trading strategies.

2.2.4 Zero-intelligence traders

Farmer et al [52] [13] develop a statistical model for the continuous double auction under the assumption of IID (independent, identically distributed) order flow. The model is referred to as zero-intelligence because of the random order flow produced. This noise trader approach differs from the traditional economics theory of perfect rationality [51]. In Part II, I adopt the noise trader model proposed in [16], which has two types of noise traders. One type places market orders; the other places limit orders. A detailed explanation is provided in section 6.2.2.

Gode and Sunder [20] present an alternative noise trader model. They present two *zero intelligence* traders referred to as *ZI-U* and *ZI-C*. The former represents an *unconstrained* model and the latter is *constrained*. For both types, each trader is provided an entitlement to buy or sell a number of units. Each of these units is associated with a particular *limit price*. Prices range from 1 to 200 units of arbitrary currency. (The values are not of importance; however, I introduce them for illustrative purpose.) ZI-U traders can place limit orders at prices anywhere within this range. As such, they are vulnerable to entering loss-making deals. ZI-C traders can only place buy limit orders at prices ranging between 1 and the limit price; the prices of sell limit orders are constrained to range between the limit price and 200. *Shout prices*, the actual prices ascribed to orders, are drawn from a uniform distribution across a trader's prescribed range. The objective of the study is to determine whether the budget constraint of the ZI-C induces trading behavior resembling the human subjects studied by Smith [53]. They find that the volatility of prices ZI-C traders falls between that of ZI-U traders (more volatile) and human traders (more stable). Further, they note that as expected from traders lacking memory or adaptation mechanisms, there is no evidence of learning with the ZI-C trader. Further, there is a slower convergence of ZI-C prices to equilibrium than observed with humans. My VLONT trader of section 7.2.2 resembles the ZI-C trader – both draw prices uniformly from a constrained range, where one boundary is dictated by a perceived value.

Cliff [12] presents a "more intelligent" zero-intelligence trader called *zero-intelligence-plus (ZIP)*. Two significant factors differentiate ZIP traders from the other noise trader models. First, ZIP traders incorporate a notion of *profit margin*. For example, a trader possessing a token with a perceived value of \$1.00 may not want to sell it unless a 20% profit is attainable. Accordingly, the trader would be willing to sell the token for \$1.20, but no less. Further, ZIP traders have an adaptive mechanism for adjusting their profit margins. If the trader is unable to sell a token with a given profit margin, the margin can be reduced to make the price more appealing to others. Conversely, if the trader is selling tokens easily, it can increase its desired profit margin with the intent of securing a greater profit per transaction. The objective of the study is to ascertain whether ZIP traders generate more human-like dynamics than do ZI-C traders. Experiments show that market prices generated by ZIP traders converge to equilibrium more rapidly than those associated with ZI-C traders. Further, it is demonstrated that the ZIP traders exhibit better allocative efficiency (defined in [54]) than do ZI-C traders. In conclusion, Cliff determines that the performance of ZIP traders in experimental markets is closer to that of human traders than is the performance of ZI-C traders.

2.2.5 Automated Trading Projects

Though not directly applicable to my research, *automated trading competitions* have gained recent popularity. Here, competitors submit software agents that trade in a simulated market environment. Agents are faced with complex tasks including bidding strategy, market prediction, and resource allocation. Perhaps, the first such competition was the Santa Fe Double Auction Tournament of 1990 [46] [47]. More recently, Wellman et al [61] [60] [56] have held widespread contests known as the *Trading Agent Competition*. Kearns et al [23] take the concept a step further with the Penn-Lehman Automated Trading Project (PLAT). The platform allows software traders to compete in a simulated market that merges the orders of the automated traders with those of a real-world, real-time limit order book (the *Island*[3] exchange).

2.2.6 Fundamentalism

The work of Chiarella and Iori [10] [9] relates to Chapter 7. They develop an order-driven market where traders use exogenously fixed rules to place both market and limit orders. The model aims to replicate dynamics observed in empirical studies of real markets: fat-tailed distribution of limit order placement from current bid/ask; fat-tailed distribution of order execution-time, fat-tailed distribution of orders stored in the order book; and long memory in the buy/sell indicator signs. Traders can place orders of varying size, as dictated by either utility maximization or a random selection process. They analyze the impact of both *fundamentalist* and *chartist* strategies on the determination of both order price and order size, the shape of the order book, the distribution of orders at distribution prices, and the distribution of order execution time. Results from the model are compared to real market data. They find that the model with risk-averse agents captures a number of stylized facts associated with double auction markets, such as fat-tailed distribution of limit order placement from current bid/ask, fat-tailed distribution of order execution-time, and fat-tailed distribution of orders stored in the order book.

[3]www.island.com

2.3 Recovery of trophic species in financial market data using community detection

In [15], Farmer develops the idea of a *market ecology*, whereby the actions of a given trading strategy affect those of other strategies. In Chapter 8, I introduce two studies that attempt to identify ecological structure in real financial markets by examining the actions of traders.

Lillo et al [31] examine the ecology of traders in the Spanish Stock Market. They investigate four different stocks (BBVA, TEF, SAN, and REP) over four independent year long periods (2001-2004). Like the London Stock Exchange, each firm represented in the data acts on the behalf of many individuals and institutions having different strategies. The data is spliced into time intervals with each interval representing an independent day of trading. For each interval, the *inventory variation* for each firm i is computed, denoted as $v_i(t)$ for day t. A correlation coefficient matrix, based on $\rho\left[v_i(t), v_j(t)\right]$ for each pair of traders i and j, is computed. The eigenvalue spectrum of the correlation matrix is computed. Using methods based on random matrix theory (RMT) [29], the largest eigenvalues are examined to test a null hypothesis stating that the inventory variation is described by uncorrelated Gaussian random variables. Lillo et al find that the first eigenvalue from the examined data is not consistent with the null hypothesis and therefore claim that the inventory variation is carrying information about the collective dynamics of firms.

Zovko and Farmer [65] examine behavioral similarity of members on the London Stock Exchange (LSE). For an individual stock, they separate each month of trading into hourly intervals. For each interval, they record whether the net trading of a given trader is positive (buying) or negative (selling). From these indicators, a correlation matrix (based on Spearman rank correlation) is computed. The largest eigenvalues of the correlation matrix are compared with a null model containing no correlations.

They conclude that for each month of trading for four stocks (AAL, AZN, LLOY, and VOD), significant eigenvalues exist that are not consistent with uncorrelated behavior of institutions. By applying a clustering algorithm to the correlations, they group institutions and find that there often exists a minority group of institutions that trades against the majority.

Part I

Tools for detecting community structure in complex networks

Chapter 3

Community detection optimization with induced fractures and merges

3.1 Introduction

In this chapter, I present a two-phase strategy for the detection of communities in complex networks. In the first phase, I simply apply any of the existing algorithms for community detection. However, in the second phase, the initial result is optimized by inducing fractures (for divisive approaches), or merges (for agglomerative approaches), in the detected communities and then optimizing the resulting constituents. I demonstrate that the two-phase approach attains consistent, high yield results. Further, in comparison to known approaches, these results can be attained without an inflation in computational time. Using the second phase alone, it is possible to incrementally improve upon an existing community detection result.

In 2002, Girvan and Newman highlighted the property of community structure in complex networks [19]. Since that time, significant literature [34] [11] [21] [14] [44] [38] [33] has concentrated on methods for detecting such sub-structure. Current approaches concentrate on optimizing *modularity*. The process yields a set of communities with each individual node of the network associated with a unique community. Because the state space of possible configurations grows exponentially with the number of nodes, optimization of modularity through exhaustive search is intractable for networks beyond a certain size. Community detection is known to be NP-complete [8].

Existing algorithms for detecting community structure are classified into two groups: divisive and agglomerative [11]. In a divisive scheme, all nodes are originally placed in a group, or community, and then recursively divided into smaller groups. An individual community is deemed indivisible when a division cannot be found that increases modularity. Communities are examined in either a breadth-first or depth-first manner until all communities are declared indivisible. Conversely, in an agglomerative approach, all nodes are initially placed in their own independent communities. Thereafter, communities are iteratively merged to form larger communities. At each step of the algorithm, the merge that yields the greatest increase in modularity is chosen. If no such increase is found, the agglomerative process halts. Figure 3.1 shows a dendrogram, a hierarchal branching diagram of the community structure for an example network, that illustrates the difference between divisive and agglomerative approaches. A divisive scheme can be regarded as top-down; whereas, an agglomerative approach can be described as bottom-up.

3.1.1 Contribution

This chapter introduces a post-detection optimization strategy that can be used in conjunction with either a divisive or agglomerative approach. The optimization algorithm has two phases. First, a traditional method is used to detect communities. Thereafter, these communities are optimized using the method of induced fractures

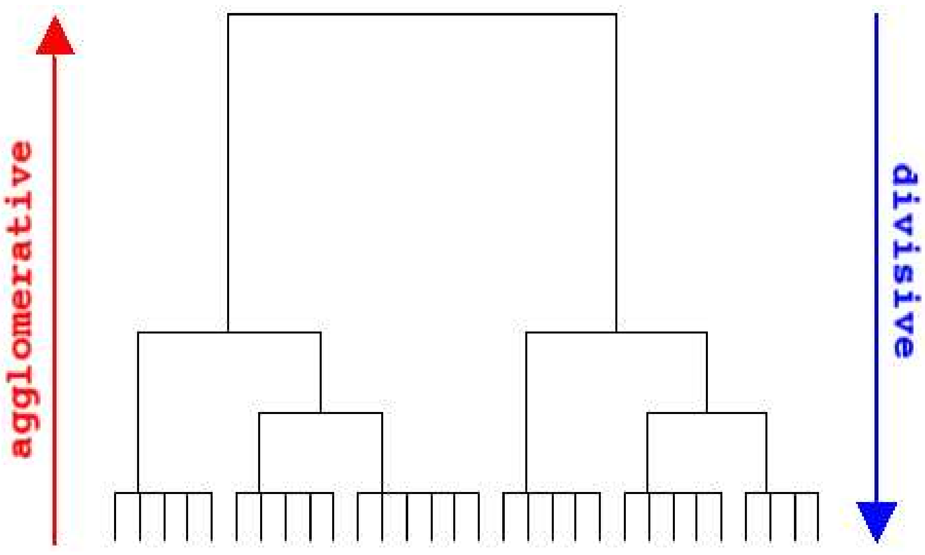


Figure 3.1: Comparison of agglomerative to divisive approach. At the bottom of the figure, each individual node of the network is represented as its own community. As the figure is traversed upwards, the joins between communities are depicted by horizontal line segments. At the top of the figure, a single line segment exists to represent a lone community. A divisive scheme uses a top-down approach to produce the dendrogram; whereas an agglomerative scheme follows a bottom up approach.

(for agglomerative approaches) or merges (for divisive approaches).

For the purpose of this chapter, I refer to the existing community detection algorithms – both the divisive and the agglomerative – as *micro-strategies*. At each step of operation, these approaches concentrate on local optimizations – either merging two communities (agglomerative) or separating an existing one (divisive). The second phase strategy introduced in this chapter operates at a higher level and uses a micro-strategy to perform its local optimizations. For this reason, I refer to the approach used in the second phase as a *macro-strategy*. The key to the strategy involves which nodes should be examined at each step. It uses a micro-strategy to perform local optimizations. However, whereas the existing micro-strategies only consider merging or splitting existing communities and preserve nodal associations, my approach allows for nodal associations to be broken and reformed. The mechanics of the macro-strategy are provided in section 3.2.1 for divisive approaches and section 3.2.2 for agglomerative approaches.

3.1.2 Simple illustrative example

Here, I provide a simple example that illustrates the advantages of my algorithm. The Zachary karate club network is small (34 nodes) and has been extensively analyzed in the micro-strategy literature. Figure 3.2 depicts the karate network. The best configuration detected prior to this work contained 4 communities with an associated modularity (mathematically defined in equation 4.7) value of $Q = 0.4188$. The dendrogram of figure 3.3 depicts how the communities were recovered by a divisive algorithm. Initially, the nodes were split into two communities. Thereafter, each of these communities was divided once more – thus, yielding four communities.

Using the same divisive algorithm as a micro-strategy, the macro-strategy detects a community configuration with an associated modularity value of $Q = 0.4917$[1]. Again, the configuration contains 4 communities. The improvement in modularity results from the migration of a single node (node 10) to a new community. In the dendrogram of figure 3.3, this migration is represented by a horizontal arrow. Note that the initial division, depicted by the top horizontal line in figure 3.3 placed node 10 on the left-hand side. The improved configuration is the result of node 10 migrating to a community located on the right-hand side. Such a migration would not be possible with existing micro-strategies; however, it is with my method of induced merges (or, fractures, for an agglomerative approach).

Figure 3.4 plots the modularity values at different stages of the algorithm (with the x-axis representing the number of communities detected at a given stage). The figure illustrates that the improved configuration, that detected by my method, re-

[1]Larger Q values indicate greater community structure.

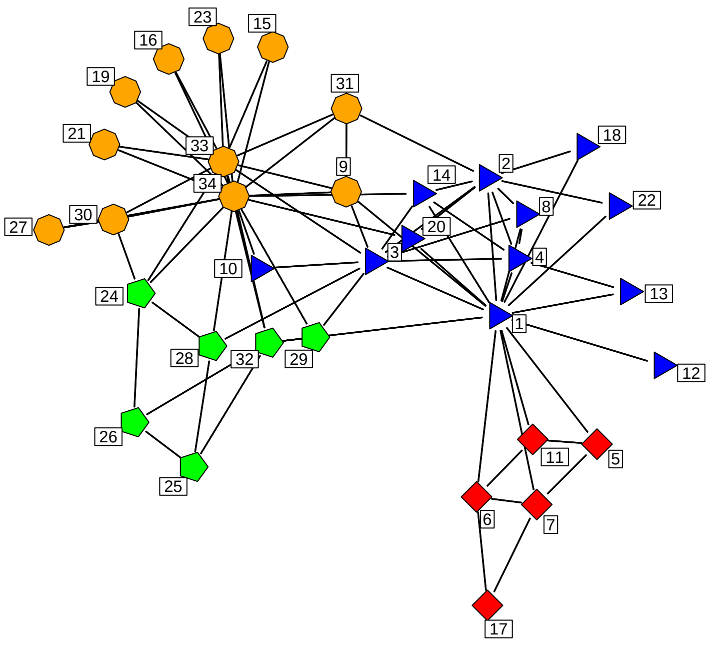


Figure 3.2: The karate club network. Nodes of each community are represented by differently shaped symbols. Edges between nodes represent friendships between members of the karate club. The best community configuration detected prior to this work, that depicted here, contained four communities and has an associated modularity value of $Q = 0.4188$.

quires a sub-optimal initial division. More specifically, had node 10 been placed in the community depicted on the right-hand side of the dendrogram in the initial division, the associated modularity value following that split would be less than that for the best configuration from the literature. In other words, if the improved configuration were to be found using an existing micro-strategy, the detection of a sub-optimal intermediate division would be required.

3.1.3 Advantages of the macro-strategy algorithm

My strategy provides several advantages: high yield, robust results, and incremental improvement. These advantages are gained without increasing the theoretical running time of the adopted micro-strategy. Table 3.1 summarizes the criteria that I will use to assess the efficacy of my strategy. Each criterion is associated with an empirical measurement. The empirical measurements assume a set of community detection results compiled on the same network – with each result based on applying the identical algorithm using a different random number generator seed.

High yield refers to the characteristic of detecting a community configuration associated with a high modularity value. In practice, yield is assessed by identifying the maximum modularity value from a set of community detection results and comparing this value to the best results from existing literature.

Robustness refers to the characteristic of an algorithm to provide consistent results – in this case, consistent modularity values. Robustness is measured by the standard deviation of modularity values over a set of community detection results.

An algorithmic approach that provides both high yield and robustness implies that the algorithm provides a consistently high yield. As such, a single application of the algorithm should provide a sufficient result. In section 3.3, I demonstrate that the macro-strategy provides both high yield and robustness.

Speed refers to the execution time of the community detection algorithm. My strategy does not increase the theoretical running time of the original community detection algorithm, and it provides both high yield and robustness with a reduced execution time from that of the established method of extremal optimization.

Incremental improvement refers to the possibility of attaining an initial result and then incrementally improving it. Current algorithms fall into one of two categories.

One category provides a quick, sub-optimal result without the option of improving it [11] [33] and the second category category [14] provides higher yield results at the expense of larger theoretical time complexities. The macro-strategy allows one to utilize an algorithm of the first category, one that provides a "quick and dirty" result, and then incrementally improve on the initial result. These improvements can be monitored and the algorithm can be halted once a sufficient result is attained.

Table 3.1: Advantages and empirical measurements for assessing the efficacy of the macro-strategy.

criterion	explanation	measures
yield	optimal detection of communities	max(Q)
robust	average detection of communities	sd(Q)
speed	length of execution	operation time
incremental	time until initial result	

The remainder of the chapter proceeds as follows. Section 3.2 describes the macro-strategy. I also introduce a new "quick and dirty" community detection algorithm, called deterministic division, that performs well with the macro-strategy. Section 3.3 gives results and comparisons using the metrics of table 3.1. Section 3.4 provides a summary and concluding remarks.

3.2 Methods

3.2.1 Divisive algorithms

The first divisive algorithm, called extremal optimization (EO) [7] [6], is known to provide high yield results [14]. Next, I introduce an algorithm called *deterministic division* (DD). After explaining the mechanics of each algorithm, I demonstrate how *induced merges* can be utilized by a divisive algorithm to optimize a community detection result.

Extremal optimization

Extremal optimization (EO) is a divisive stochastic algorithm that has optimizes modularity for the task of community detection. Initially, the nodes are randomly assigned to one of two partitions. After the initial modularity is computed, a single node is migrated from one partition to the other, and modularity is recomputed.

A counter, denoted K, tracks the number of moves since the last modularity improvement. If modularity fails to improve following a migration, the counter is incremented. Otherwise, the counter is reset to zero, and the partitioning is recorded along with its associated modularity value. This partitioning represents the best detected configuration. The process continues until the counter reaches a predetermined threshold. For each division, the size of the original community, denoted $|C_i|$, determines the stopping criterion such that the maximum allowable number of steps is $S = \alpha|C_i|$ ($\alpha = 3$ in the experiments of section 3.3). Once the counter reaches the threshold, such that $K = S$, the process terminates and the best detected configuration is retained. If the split has improved the modularity value, the global set of communities is updated to reflect the division. Otherwise, it remains unchanged and the original community is marked indivisible.

An important component of the EO algorithm is choosing which nodes to migrate. Rather than choosing nodes at random, I associate a value ΔQ^u (see equation A.1 of appendix A.1) with each node u. This method resembles hill-climbing used in other settings and biases the search towards immediate improvements. The value ΔQ^u represents the change in modularity that occurs by migrating the specified node. This approach differs slightly from that of [14], which uses a heuristic to approximate ΔQ^u.

A list containing the ΔQ^u values is maintained. The list of ΔQ^u values is ranked, and a node is probabilistically chosen using a method known as τ-EO [14] [7]. Using this process, a node of rank q is chosen with probability of $P(q) \approx q^{-\tau}$ where $\tau = 1 + \frac{1}{\log |C_i|}$. Note that the process of ranking the nodes costs $O(nlogn)$, where $n = |C_i|$.

After each migration, all ΔQ^u values need to be updated. Rather than recompute the value for each node, it is more efficient to compute the change in ΔQ^u induced by migrating a node m, denoted as $\Delta^2 Q^{um}$ (see appendix A.1 for a mathematical description). The computational cost of computing $\Delta^2 Q^{um}$, $O(1)$, is less than that for ΔQ^u, which is $O(|C_i|)$.

Deterministic division

I introduce the method of deterministic division (DD) as an alternative to EO. On its own, DD falls into the category of *quick and dirty* community detection algorithms [11] [33]. However, DD performs well when paired with the macro-strategy.

DD differs from EO in the manner nodes are selected for migration. Whereas EO chooses nodes probabilistically, DD always chooses the node with the highest ΔQ^u value. If more than one node shares the highest ΔQ^u value, a single node from the subset is chosen. The algorithm halts when there no longer exists a $\Delta Q^u > 0$.

For an individual migration, DD is less expensive than EO. As noted, after a migration, the ΔQ^u values need to be updated. This process costs $O(n)$. In DD, it is merely necessary to identify the best ΔQ^u value prior to the next migration. However, with EO, it is necessary to rank the entire ΔQ^u list. The ranking process costs $O(nlogn)$. In section 3.3, I will demonstrate that the macro-strategy using DD is capable of achieving similar results to the macro-strategy using EO, but in reduced computational time.

Induced merges for divisive approaches

The first phase of the macro-strategy yields a set of communities. The second phase, the post-detection phase, allows for nodes to be juggled amongst these communities. For divisive algorithms, this process involves merging two communities and then dividing the conglomerate, as illustrated in figure 3.5.

During the initial phase of the algorithm, once nodes are separated into different communities, they may never be recombined. As demonstrated in section 3.1.2, it may be the case that the separation of two nodes positively contributed to an intermediate step of the algorithm, yet negatively contributes to global optimization. The mechanics of the initial phase do not allow for recovery from such a deleterious step. By inducing the merge of two communities and then dividing the conglomerate, nodes that were formerly separated have an opportunity for recombination. These potential recombinations allow for further global optimization.

Given $|C|$ communities, an exhaustive search of all possible community recombinations yields $\frac{|C|\dot{(}|C|-1)}{2}$ induced merges. The first phase of the algorithm requires only $|C| - 1$ divisions. Thus, an exhaustive search of the possible merges entails a large computational penalty. Ideally, the second phase of the algorithm should improve modularity without increasing the running time. The initial *operation time*, denoted as T_1, represents the number of steps in the first phase. The operation time of the second phase is denoted as T_2. A parameter provided to the algorithm, denoted as ρ, dictates the ratio of operation time allowed for the second phase of the algorithm as compared to that established in the first phase. When $T_2 \geq \rho T_1$, the second phase of the algorithm is halted. In these experiments, I set $\rho = 10$ for EO experiments and $\rho = 50$ for DD experiments. This ensures that the overall running time of the algorithm increases by only a constant multiplier. As a result, the big-O running time remains unaffected.

The pseudo-code provided in Algorithm 1 describes the second phase. The `while-loop` is entered and continues while $T_2 < \rho T_1$ (line 3). A list of all possible community pairs is required. Initially, this list is empty (line 4) and thus it is created (line 5). A community pair is selected and then removed from the list (line 7). The pair of communities is merged (line 8) and then divided using a micro-strategy (line 9). The fitness (modularity, in this case) resulting from the division is compared to the pre-division fitness (line 10). If the fitness improves, the new community alignments are applied (line 11) and the fitness is updated (line 12). Further, the list of community pairs is updated (line 13) such that any possible pair that includes one of the altered communities is appended (given that it was previously removed). Each function called within the `while-loop`, increments T_2 (once for each operation performed). If the list of community pairs is exhausted (line 4) before the `while-loop` terminates, it is regenerated (line 5).

3.2.2 Agglomerative methods

The agglomerative method used in this chapter is the CNM algorithm [11]. Initially, all nodes are placed into independent communities. The values of ΔQ_{ij} for all connected communities i and j are calculated. The largest value for each community i is placed into a max-heap. At each step, the largest ΔQ_{ij} value from the max-heap is selected and the communities are i and j are combined. The modularity is updated such that $Q_{t+1} = Q_t + \Delta Q_{ij}$ and the max-heap is updated. The process continues until only one community remains. At each step, if a community configuration with a higher modularity value is encountered, it is recorded. The best identified configuration is returned as the result. The theoretical running time of the CNM algorithm is $O(2md \log n) = O(md \log n)$, where n represents the total number of nodes, d represents the depth of the resulting dendrogram, and $2m$ is the total degree in the network.

Algorithm 1 Pseudo-code for the second phase of the macro strategy. T_2 is incremented within each function nested in the `while-loop`.

```
1: $T_2 \leftarrow 0$
2: $fitness \leftarrow fitnessFromFirstPhase()$
3: while $T_2 < \rho T_1$ do
4:     if $isEmpty(pairsOfCommunities)$ then
5:         $listOfCommunityPairs \leftarrow createAllPossibleCommunityPairs()$
6:     end if
7:     $communityPair \leftarrow choosePairFromListAndRemove()$
8:     $mergedCommunity \leftarrow mergeCommunityPair()$
9:     $[newFitness, newCommunityPair] \leftarrow divideUsingMicroStrategy()$
10:    if $newFitness > fitness$ then
11:        $applyNewCommunityPairAlignment()$
12:        $fitness \leftarrow newFitness$
13:        $appendCommunityPairList()$
14:    end if
15: end while
```

Induced fractures for agglomerative approaches

Figure 3.6 illustrates the process of an induced fracture using an agglomerative process. The post-optimization method employs a series of induced *fractures*, whereby two existing communities are fractured into their nodal constituents. For example, given two communities, community C_i and community C_j, the fracture yields $(|C_i| + |C_j|)$ independent communities, each containing a single node. Thereafter, the agglomerative algorithm is applied to this subset of nodes – attempting to find a community configuration that yields a higher modularity value than associated with the pre-fracture configuration. If an improvement is found, the communities are updated to reflect the change.

As described in section 3.2.1, the goal of the post-optimization phase is to improve modularity without increasing the theoretical running time of the original algorithm. As such, an exhaustive search of all possible fracture pairs is unreasonable. Therefore, I adopt the same approach as previously outlined. The operation time from the initial phase, denoted as T_1, is recorded. The second phase terminates when $T_2 \geq \rho T_1$, where T_2 is the cumulative operation time of the second phase and ρ is a supplied parameter. For the experiments of section 3.3, I set $\rho = 30$.

3.3 Results

To examine the efficacy of the macro-strategy, I independently employ each of the micro-strategies introduced in section 3.2 within the macro-strategy and examine five networks that have been extensively examined in previous literature. The size of the networks graduates from 34 nodes for the Zachary karate club to 27519 nodes for the condmat network that describes the collaborations of physicists. For each network, I apply the macro-strategy using a specified micro-strategy 50 times, using a different random number generator seed for each application. I independently assess the characteristics of high yield, robustness, and execution speed.

3.3.1 High yield

Here, I demonstrate the characteristic of *high yield* for the post-detection optimization techniques of section 3.2. For each algorithm – extremal optimization (EO), deterministic division (DD), and CNM agglomerative (CNM) – I compute two indicator ratios for each of the five networks. The M indicator, computed independently for each algorithm type, compares the mean modularity value following the second phase to that following the first phase. Mathematically, $M = \bar{Q_2}/\bar{Q_1}$, where $\bar{Q_1}$

represents the mean value over the set of modularity values following the first phase, and $\bar{Q_2}$ represents the mean value over the set of modularity values following the second phase. If $M > 1$, on average, the second phase improves the community configuration discovered in the first phase. Otherwise, the second phase fails to improve the initial result.

The second indicator computed for each algorithm is $X = max(Q_2)/max(Q_{LIT})$, where $max()$ returns the maximum value and $max(Q_{LIT})$ represents the best modularity value for the specified network from previous literature. A value of $X > 1$ indicates that my algorithm improves upon the best result found in existing literature. Table A.4 in appendix A.4 compares results from previous literature.

Table 3.2 summarizes the M and X indicators (as well as others to be explained later) for the EO, DD, and CNM agglomeration algorithms for the five networks. The first four columns relate to the EO algorithm. The middle three columns relate to the DD algorithm, and the final three columns relate to the CNM agglomeration algorithm. For now, I concentrate on the first two columns of each section, those labeled M and X.

First, observe the M values. For each of the algorithms, $M > 1$ for all networks. These values demonstrate that the second phase consistently improves upon the results detected by the first phase alone. Note that the DD and CNM indicator ratios are greater than that for the EO algorithm – especially for the larger networks. Recall that the DD and CNM algorithms fall into the category of "quick and dirty" algorithms; whereas, EO does not. As a result, the results associated with these algorithms show greater improvement during the second phase.

Secondly, with the exception of the jazz network, $X > 1$ for each algorithm on all networks. Thus, by incorporating the second phase, my approach consistently detects results that exceed those established in previous literature. The only other exception is where $X_{DD} = 0.9956$ for the PGP network. The modularity value detected by DD ranked second when compared to the existing literature. In section 3.3.3, I will show that DD is able to achieve this result in approximately one-tenth of the time required by EO. It is possible that this DD result could improve by extending the length of the second phase (increasing ρ).

Table 3.2: Comparison of ratio indicators for extremal optimization (EO), deterministic division (DD), and CNM agglomeration (CNM).

network	M_{eo}	X_{EO}	N_{EO}	S_{EO}	M_{DD}	X_{DD}	N_{DD}	M_{CNM}	X_{CNM}	N_{CNM}
karate	1.0176	1.0024	1.0024	0.0000	1.0589	1.0024	1.0024	1.1019	1.0024	1.0016
jazz	1.0127	0.9999	0.9996	0.0400	1.0242	0.9999	0.9994	1.0127	0.9993	0.9984
celegans	1.0724	1.0484	1.0421	0.2549	1.1542	1.0387	1.0280	1.1019	1.0420	1.0308
email	1.0518	1.0156	1.0119	0.1513	1.1212	1.0121	1.0065	1.1377	1.0068	1.0000
pgp	1.0462	1.0256	1.0228	0.1978	1.1120	0.9956	0.9873	1.0151	1.0175	1.0087

3.3.2 Robustness

To demonstrate the robustness introduced by the second phase of my algorithm, I examine two additional indicators. The first, referred to as S, compares the consistency of modularity results following the second phase to those following the first phase. Mathematically, $S = sd(Q_2)/sd(Q_1)$, where $sd()$ returns the standard deviation. Recall that robustness refers to the attribute of attaining consistent results over a range of seeds. Standard deviation measures the consistency of results. When comparing two standard deviations, a lower value indicates less variation and greater consistency. Thus, $S < 1$ demonstrates that the second phase achieves more consistent results.

I do not compute S for the deterministic algorithms – DD and CNM agglomeration. Because of their deterministic nature, results for these algorithms do not vary during the first phase. Stochastic implementations of these algorithms are possible, but not used in this work. Results for these algorithms vary following the second phase because of the stochastic process used to choose pairs of communities for each fracture or merge.

The fourth column of table 3.2 provides the S values for the stochastic EO algorithm. For each of the five networks, $S < 1$. The second phase optimization lowers the standard deviation in each case. This result establishes that the macro-strategy achieves robustness.

To further demonstrate robustness and high yield, I introduce the indicator $N = \bar{Q_2}/max(Q_{LIT})$, which compares the the mean modularity following the second phase to the best result from existing literature. I have demonstrated, with the X indicator, that introduction of the second phase improves upon the best established results of existing algorithms. The N indicator examines whether, on average, the established values are exceeded. Examination of table 3.2 shows that N displays the same pattern as X. For each algorithm/network pairing such that $X > 1$, the same is true for N. This result demonstrates that the macro-strategy provides consistently high yield results.

3.3.3 Speed

It is natural to assume that the incorporation of a second phase to the algorithm will lead to an execution time penalty. My goal is to demonstrate that by using a "quick and dirty" micro-strategy paired with the second phase optimization, a reduction in execution time can be established. Towards this end, I compare the time consumed by the macro-strategy using the DD algorithm to that using the EO algorithm.

Three definitions of time are examined: operation time, merge time, and migration time. *Operation time* refers to the number of fundamental computational operations performed. *Migration time* refers to the cumulative number of node migrations. *Merge time* refers to the number of community merges during the second phase.

Figure 3.7 presents results for the email network. In each graph, the curve represented with inverted triangles represents the DD algorithm and the curve with circles represents the EO algorithm. Further, the y-axis of each graph represents the mean modularity value over all seeds at the indicated time. The x-axis represents time: operation time on top, migration time in the middle, and merge time on the bottom. The starting x-coordinate for each EO curve indicates the mean start time for the second phase. The dashed vertical line in each graph indicates the time at which the EO curve modularity value equates with the maximum modularity value attained by DD.

With reference to the top graph of figure 3.7, it demonstrated that, on average, it requires an approximate factor of 9 increase in operation time for EO to match the maximum modularity value attained by DD. The middle graph illustrates that EO requires approximately twice as many migrations to attain the maximum modularity found by DD. The bottom graph shows that DD performs approximately 11 times as many merges than does EO to reach the maximum DD modularity value.

Table 3.3 summarizes these time ratios for each network. In each case, the ratio compares the time required by EO to that of DD to reach the maximum DD modularity value. In each case, DD requires less operation time. However, on larger networks, DD actually performs more migrations than does EO. Recall that an individual migration costs less in DD than it does in EO. Thus, it is possible for operation time and migration time ratios to vary inversely. For each network, DD performs more merges – with this relationship positively correlated with the size of the network. Because individual DD migrations are less expensive, the algorithm is able to explore a greater number of merges than EO without incurring more operation time.

Table 3.3: Time ratios of EO to DD. The computed ratio compares the time at which EO reaches the maximum modularity value attained by DD to the DD time required.

network	operation time	migration time	merge time
karate	7.09	5.45	0.489
jazz	3.37	1.55	0.174
metabolic	5.11	1.18	0.043
email	8.99	1.99	0.093
pgp	9.93	0.61	0.028

3.4 Summary and conclusions

In this chapter, I introduced a two-phase methodology for detecting community structure in complex networks. The first phase can be performed by any of the community detection algorithms from existing literature. In the second phase, I utilize induced fractures and merges of the existing communities to optimize the result established in the first phase.

In section 3.3 I demonstrated that the macro-strategy provides high yield, robust results. As a result, even with a single application of the method, a practitioner can be confident with the result. Furthermore, when using a "quick and dirty" micro-strategy such as DD or CNM agglomeration, the macro-strategy performs faster than the known strategy EO while providing similar or better results.

A major advantage of the second phase is that of incremental improvement. The method makes it possible to take an existing community detection result and improve upon it. Furthermore, progress of the algorithm can be tracked and terminated once a sufficient result is attained.

In this chapter, I independently examined the two divisive (EO and DD) algorithms and the agglomerative algorithm (CNM). However, it is possible to "mix and match" micro-strategies within the context of the macro-strategy. When using a traditional approach and not utilizing the the second phase, it has been demonstrated that the EO micro-strategy provides better results than do DD and CNM. It would be possible to execute the macro-strategy using one of the "quick and dirty" micro-strategies until the modularity value appears to asymptote. Thereafter, EO (or another high yield approach such as [39]) could be adopted as the micro-strategy to potentially optimize modularity further.

As an implementation detail, for the second phase, I employed a stochastic process to choose pairs of communities for the induced fractures/merges. It could be the case that a probabilistic selection process will yield faster improvements in modularity for particular network structures. For instance, if the sizes of the underlying communities of a network were to be power-law distributed, it might be more efficient to choose communities probabilistically based on size. As such, larger communities would likely be exposed to more optimization stages (divisions or merges) early in the computation. The selection process warrants further examination.

Dendrograms provide a valuable tool for evaluating the hierarchal community structure of a network. It is straightforward to construct a dendrogram when using a traditional micro-strategy – each division or merge represents an intersection of branches. However, the induced fractures/merges of the second phase corrupts the structure of the dendrogram resulting from the first phase of the macro-strategy. It is possible to reconstruct a valid dendrogram from the result of the second phase. Each community can be collapsed into a super-vertex. Thereafter, using the CNM agglomeration algorithm, super-vertices can be merged until a single community remains. A trace of the merged communities provides the blueprint for a dendrogram.

The two-phase method proposed in this chapter can be used in conjunction with any of the micro-strategies from existing literature. By utilizing the second phase of induced fractures/merges, it is possible to improve any of the existing divisive or agglomerative micro-strategies. This benefit comes at the expense of additional computation. However, as demonstrated, incorporation of the second phase allows a practitioner to utilize light-weight “quick and dirty” micro-strategies to establish competitive results with a reduction in computation.

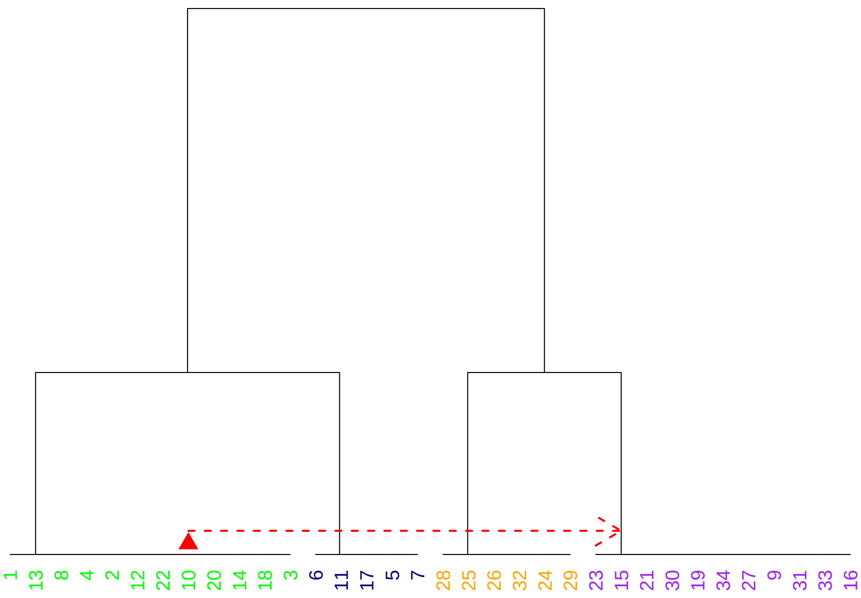


Figure 3.3: Karate club dendrogram. The dendrogram represents how the nodes were divided to detect the four communities representing the best community configuration from previous literature ($Q = 0.4188$). Using the macro-strategy, a configuration with $Q = 0.4197$ was detected. The increase in modularity results from node 10 migrating to a new community. This migration is represented by a horizontal arrow. Note that the initial division, at the top of the dendrogram, delegated node 10 to the left-hand side. The improved configuration results from migrating node 10 to a community on the right-hand side. Such a migration would not be possible with existing micro-strategies.

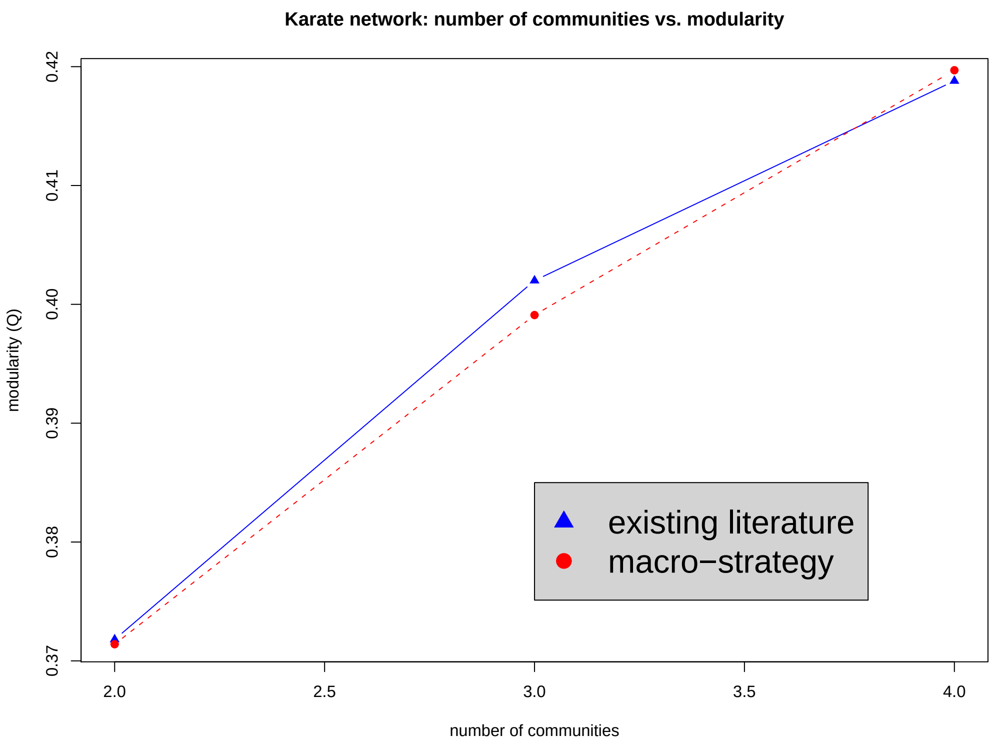


Figure 3.4: Comparison of modularity values for karate network. Modularity is represented on the y-axis. The x-axis represents the stage of the algorithm with the number of communities detected at that point. The best result from previous literature (Q=0.4188) is represented by a solid curve. The improved result (Q=0.4197) is represented by a dashed line. With regards to macro-strategy result, I present the configurations that would have existed at each stage had node 10 initially been placed on the right-hand side of the dendrogram.

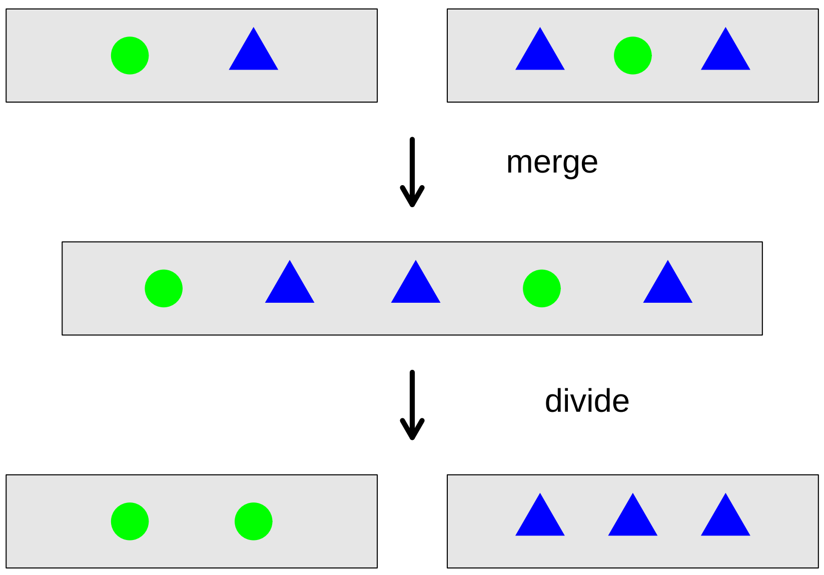


Figure 3.5: Illustration of an induced merge for a divisive approach. Nodes of one communities are depicted with circles, the other with triangles. First, the communities are merged into a single entity. Then, the divisive algorithm is applied to the merged community. In this example, the division yields two new communities. If the modularity value associated with the new configuration exceeds that of the original, the new configuration is enforced and the modularity is updated accordingly.

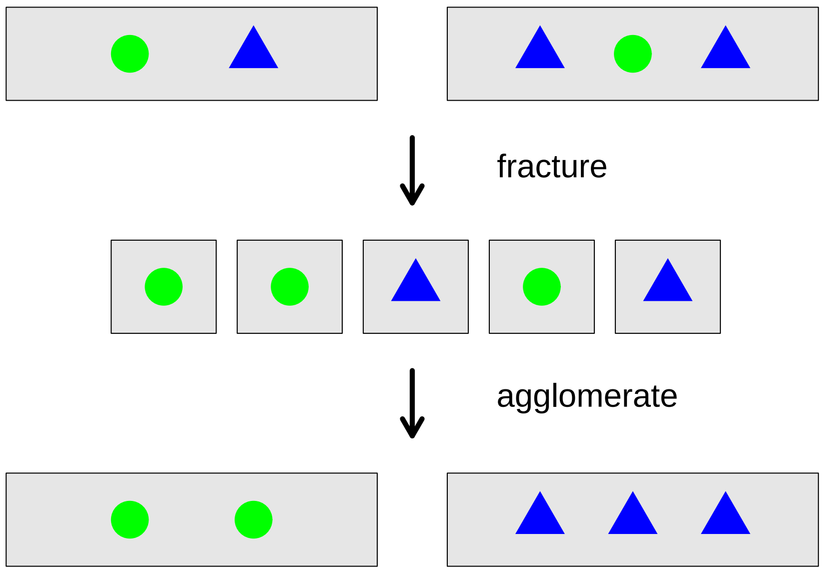


Figure 3.6: Illustration of an induced fracture for an agglomerative approach. Nodes of the original communities are depicted with either a circle or triangle. After the fracture, each node represents an independent community. The agglomerative process then combines the nodes to yield new communities. In this example, the resulting communities are different than the original. If the modularity associated with the new configuration exceeds that of the original, the new configuration is enforced and the modularity value is updated accordingly.

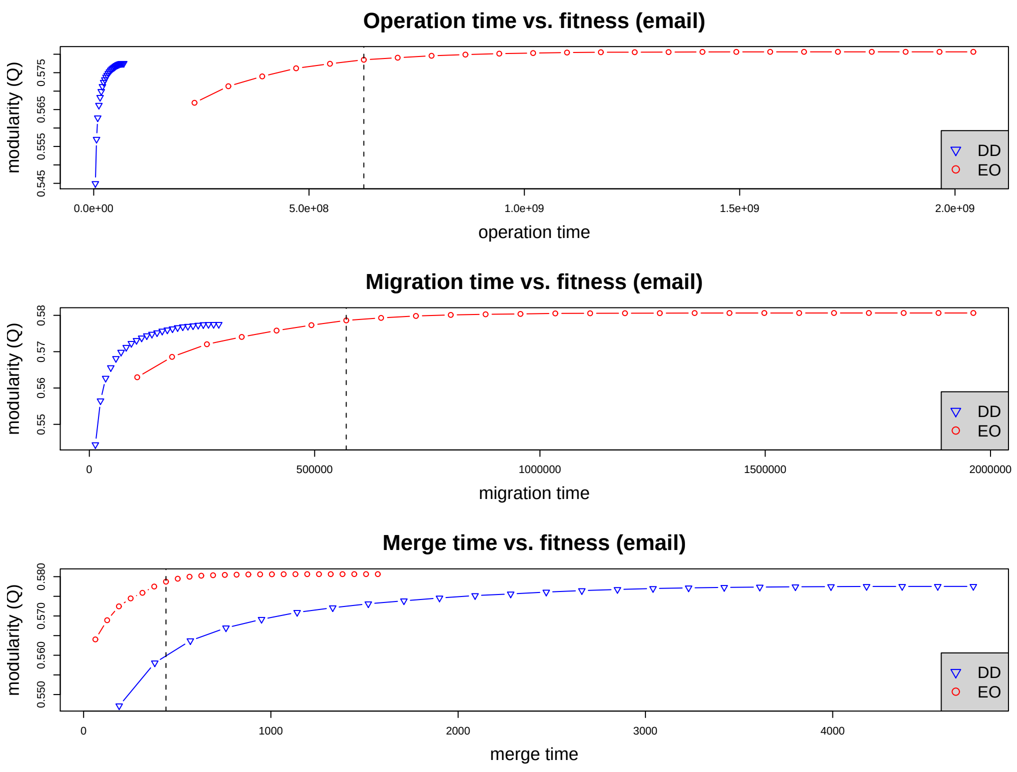


Figure 3.7: Comparison of modularity versus time for the email network. The y-axis of each graph represents the mean modularity value over the set of community detection results. The x-axis differs in each graph – operation time on the top, migration time in the middle, and merge time on bottom. DD results are represented by inverted triangles. EO results are represented by circles. The dashed horizontal line in each plot indicates the time at which EO attains the maximum modularity value attained by DD. The x-axis starting point of each EO curve approximates the starting point of the macro-strategy second phase.

Chapter 4

Divisionality: a measure for fine granularity community structure

4.1 Introduction

The majority of literature regarding community detection within complex networks has focused on optimization of *modularity*, which allows one to quantify the success of a partitioning. In this chapter, I experimentally demonstrate that modularity suffers from a *resolution limit*, which was independently demonstrated through analysis by [18]. Because of this limit, optimization of modularity is only able to detect coarse-granularity community structure. I introduce a new measure of community structure, called *divisionality*, that allows for the detection of fine-granularity community structure. The efficacy of the two measurements is demonstrated and compared on graphs with known structure.

Modularity is merely one way to assess community structure. The task of identifying communities is not a clear cut endeavor. Consider defining the academic communities within a university. One could identify communities as they are determined by departments: the English department community, the Physics department community, and so on. However, communities often follow a hierarchic structure. Consider a typical Computer Science department. Within this discipline, there generally exist groups specializing in either hardware, software, or theory. In this example, partitioning at the level of the departments represents a *coarse-grained* level of community structure. In contrast, the groups defined at the level of specialties within a department represents a *fine-granularity* community structure partitioning.

In this chapter, I experimentally demonstrate the resolution limit inherent to modularity. By optimizing modularity on a given graph, only coarse-granular communities can be detected. It is possible to isolate finer-granularity community structure through optimization of modularity; however, this requires isolating sub-networks associated with communities of the original network and then recursively optimizing modularity on these modules. A drawback of this modularity-based recursive approach is that one cannot compare the modularity values of two independent partitioning of the same network to assess which is better. Modularity will declare the fine-granular communities sub-optimal. The alternative measure introduced in this chapter, called *divisionality*, overcomes the resolution limit and allows for detection of fine-granular structure. Further, optimization of divisionality provides a self-contained method for detecting such structure – it is unnecessary to isolate sub-networks and recursively optimize the measure. A final advantage of divisionality is that it provides a global measure of community structure that allows for the comparison of different fine-granular community partitionings on the same network.

4.1.1 An example

Here, I illustrate the resolution limit inherent in modularity. Figure 4.1 depicts a network with known structure: the 2005 Major League Baseball (MLB) schedule.

Each node in the network represents a team. Each weighted edge indicates the number of games played between a pair of teams. Edge weights contribute to both modularity and divisionality. The higher the edge weight, the stronger the communal strength between the associated nodes.

Before examining figure 4.1 further, I explain the structure of MLB and its schedule. MLB consists of thirty teams split into two leagues. The American League (AL) consists of fourteen teams; the National League (NL) is comprised of sixteen teams. Within each league, teams are split into three divisions – yielding 6 divisions for the entirety of MLB. Each team plays 162 games during the season. The schedule for the MLB season fosters rivalries by imposing what is known as an *uneven* schedule. The uneven schedule has teams playing more games against their division rivals than other teams (between $33 - 48$ percent of the 162 games depending on the division). Furthermore, of games not played against division rivals, the large majority are played against other teams from the same league. Interleague games are a rarity – comprising approximately ten percent of each team schedule. The MLB schedule is of interest because it contains known, regular structure.

With reference to figure 4.1, the network contains two different sets of nodes, each being represented by a different color. The sets represent the two communities recovered by optimizing modularity on the network using the deterministic division algorithm presented in section 3.2.1 (with $\rho = 50$). The nodes in the core of the network (diamond shaped), represent teams from the AL; the nodes on the periphery (rectangular) represent teams from the NL. The communities recovered through modularity optimization match that of the known league split. For the sake of visual clarity, only edges attached to at least one of the AL teams are shown; edges connecting two NL teams have been removed. The centrally located bold edges represent those of the highest weight. With regards to the MLB schedule, these edges are known to exist between teams residing in the same division. The dotted edges represent the edges with the lowest weight – the interleague games. Edges of medium weight are represented by thin, solid lines.

The partitioning provided by modularity, that of the two leagues, is clearly depicted in the figure 4.1. All of the edges of medium to high weight (all non-dotted edges) exist between teams of the AL (within the core of the network). All edges between the two communities (that of the core nodes and the peripheral nodes), are of the lowest weight (dotted edges). Modularity succeeds in identifying a coarse-grained partitioning within the network – that existing at the league level. However, it is known that there exists underlying community structure in the MLB network not identified by modularity: the divisional structure. The leagues consist of approximately fifteen teams each. The divisions consist of approximately five teams apiece. Thus, I regard the divisional structure to be more fine-grained. Divisionality identifies community structure at the divisional level.

4.2 Methods

In this section, I mathematically define divisionality and compare it to modularity. Both measures require a network and a set of communities, denoted $\{C\}$, as input. Each community $C_i \in \{C\}$ represents a set of nodes. Each node is a member of a single community.

4.2.1 Mathematical definition of divisionality

Divisionality, D, is the weighted average of a local divisionality ratio D_i computed on each community. For simplicity of description, I assume an unweighted[1] network where $e_{rs} \in \{0, 1\}$ edge weight between nodes r and s. I define D_i as:

[1]The mathematics apply to weighted networks as well.

$$D_i = \frac{\frac{w_{ii}}{T}}{(\frac{a_i}{T})^2} \tag{4.1}$$

$$= \frac{w_{ii}T}{a_i^2} \tag{4.2}$$

.

Here, $\{w_{ii} = \sum_{rs} e_{rs} | r \in C_i, s \in C_i\}$ represents twice the number of edges in which both ends terminate at nodes belonging to community i. Further, $a_i = \sum_j w_{ij}$ provides the sum of all edge weights with at least one end attached to a node residing in community i, and $T = \sum_i \sum_j w_{ij}$ is twice the total number of edges in the network.

The numerator of equation 4.1 represents the ratio of edges encapsulated by community C_i. The denominator provides a null test. The fraction a_i/T represents the expectation that one end of a given edge will belong to community C_i. By squaring this fraction, the expectation that both ends of a given edge reside in community Ci is established. Thus, the ratio of 4.1 compares the actual edge weight encapsulated by community C_i to the expectation for this value. A value of $D_i > 1$ indicates that the edge weight encapsulated by community C_i is greater than expected. The higher the value D_i, the greater the community structure exhibited by community C_i. Conversely, if $D_i < 1$, the observed edge weight encapsulated by community C_i is less than expected and indicates a lack of community structure.

Using D_i, the global divisionality measure can be expressed as:

$$D = \frac{1}{N} \sum_{C_i \in \{C\}} \frac{|C_i|}{N} D_i \tag{4.3}$$

where N represents the total number of nodes and $|C_i|$ represents the number of nodes in community C_i. The internal ratio $|C_i|/N$ weights the contributions of each D_i term by the relative size of the community C_i. The D_i terms of larger communities are emphasized more than those of smaller communities. The $1/N$ ratio outside of the sum normalizes the measure to indicate the community strength per node. Although it is not valid to compare divisionality measures from different networks, this normalization process provides for an intuitive basis comparison.

Combining equations 4.1 and 4.3, we establish the full mathematical description of divisionality:

$$\begin{aligned} D &= \frac{1}{N} \sum_{C_i \in \{C\}} \frac{|C_i|}{N} D_i && (4.4) \\ &= \frac{1}{N} \sum_{C_i \in \{C\}} \frac{|C_i|}{N} \frac{w_{ii} T}{a_i^2} && (4.5) \\ &= \frac{T}{N^2} \sum_{C_i \in \{C\}} |C_i| \frac{w_{ii}}{a_i^2} && (4.6) \end{aligned}$$

4.2.2 Mathematical comparison to modularity

In section 4.3, I present results and make consistent reference to modularity measures and compare them to those of divisionality. For comparison, using similar notation as in section 4.2.1, I present the mathematical definition of modularity in equation 4.7.

$$Q = \sum_{C_i \in \{C\}} \frac{w_{ii}}{T} - (\frac{a_i}{T})^2 \qquad (4.7)$$

The mathematical measures of divisionality and modularity are similar. Both are based on the same null test: the $\frac{a_i}{T}^2$ term. However, while divisionality is based on the ratio of $\frac{w_{ii}}{T}$ to the null test value, modularity computes the difference. A value of $Q > 0$ indicates the presence of community structure; whereas, $Q \leq 0$ indicates a lack of it. Unlike divisionality, modularity does not weigh the individual contributions of the terms inside of the summation. Because divisionality is based on a ratio (of observed to expected encapsulated edge weight) as compared to a difference, it tends to drive deeper into the community structure hierarchy than does modularity.

4.3 Results

In this section, I compare the results of applying the community structure measures of modularity and divisionality to six different networks of known structure. Each example is based on the schedule of a sports league. Teams are represented by nodes. Games between teams are represented by weighted edges. The weights of the edges indicate the number of games played between the two associated teams. The first four networks are based on schedules from professional sports: Major League Baseball (section 4.3.1), the National Football League (section 4.3.2), the National Basketball Association (section 4.3.3), and the National Hockey League (section 4.3.4). Each of these leagues is comprised of approximately thirty teams. The final two networks are based on college sports: NCAA Football (section 4.3.5) and NCAA Basketball (section 4.3.6). These networks consist of 119 and 324 teams respectively.

The results discussed throughout this section are presented in table 4.1. For each network, the actual number of teams, divisions, and leagues are presented. In addition, for each case, divisionality and modularity values are compared. For each measure, three values are provided: Q_{opt}, Q_{gov}, and $|C_Q|$ for modularity and D_{opt}, D_{gov}, and $|C_D|$ for divisionality. With regards to the divisionality statistics, D_{opt} represents the best value found by optimizing divisionality on the network, D_{gov} represents the divisionality value associated with the alignments imposed by the governing body of the sports association, and $|C_D|$ indicates the number of communities associated with the D_{opt} value. The Q values are defined similarly, but with refer to modularity rather than divisionality. I used the my deterministic division algorithm ($\rho = 50$) to conduct all of the optimizations.

Table 4.1: Comparison of divisionality and modularity values

	league composition			divisionality			modularity		
data set	teams	leagues	divisions	D_{opt}	D_{gov}	$\|C_D\|$	Q_{opt}	Q_{gov}	$\|C_Q\|$
MLB	30	2	6	.0887	.0887	6	.39	.28	2
NFL	32	2	8	.0941	.0941	8	.38	.25	4
NBA	30	2	6	.0423	.0377	2	.13	.02	2
NHL	30	2	6	.0590	.0590	6	.28	.13	2
CFB	119	12	17	.0778	.0770	19	.65	.51	9
CBB	324	32	36	.0651	.0616	51	.618	.546	14

4.3.1 Major League Baseball (2005)

As described in section 4.1.1, Major League Baseball (MLB) is comprised of thirty teams that are split into two leagues. Within each of these leagues, teams are divided into three divisions, with the sizes ranging between four and six teams. Each team plays 162 games during the season, with one third to one half of these games being against division rivals. Only ten percent of the games pit teams residing in different leagues. Figure 4.2 compares the schedule composition of the various leagues discussed in this chapter according to the approximate percentage of games played against three types of opponents: division rivals, league rivals not in the same division, and inter-league (or inter-conference) opponents.

With reference to table 4.1, it is demonstrated that optimization of modularity bisects the MLB network, with each partition representing one of the leagues (as shown in figure 4.1). The bisection is associated with a with a modularity value of $Q_{opt} = 0.39$. In contrast, the modularity value for a network partitioned by the actual MLB divisional alignment results in a modularity value of $Q_{gov} = .28$. Given that $Q_{opt} > Q_{gov}$, it can be said that modularity does not favor the MLB divisional structure as the optimal community structure. In contrast, divisionality recovers six partitions, with each representing an MLB division. With reference to table 4.1, $D_{opt} = D_{gov}$, thus implying that optimization of divisionality accurately reflects the official MLB divisional structure.

Figure 4.3 depicts the structure of the divisionality partitioning. Teams of each independent division are represented by a unique shape. The diamond-shaped nodes in the center of the network represent the teams of the *NL East* division (Braves, Mets, Phillies, Marlins, and the Washington Nationals). For visual clarity, only edges connected to this division are shown. The dotted edges represent edges of the lowest weight and represent inter-league games. The dashed edges represent edges of medium weight. With the exception of those connected to *AL East* teams (Yankees, Red Sox, Devil Rays, Orioles, and Blue Jays), all of these middle weight edges represent intra-league matchups with teams in different divisions. The bold edges in the core of the network represent those with the greatest weight (sixteen to eighteen games). All of these occur between teams of the *NL East*. Comparison of figures 4.3 and 4.1 illustrates a difference between divisionality and modularity. Divisionality identifies concentrations of nodes that encapsulate edges of the greatest weight. In contrast, modularity isolates groups of nodes connected by all but the weakest of edge weights.

Figure 4.4 provides an alternative depiction of the difference between divisionality and modularity. The tree structure, referred to as a dendrogram, depicts the community splits that occurred during the optimization of divisionality. At the top of the tree (where the y-axis is indicates $height = 6$), all nodes are contained in one partition. At the bottom of the tree ($height = 0$), the nodes are color coded according to the community with which they are eventually associated. The forks in the middle of the tree indicate how the partitions were split. At $height = 2$, the network is split into two partitions, with each representing one of the MLB leagues. This level in the tree accords with the termination point for modularity optimization. Whereas modularity stops at $height = 2$, divisionality continues to dissect the network. At $height = 1$, one of the three divisions in each league is cleaved from the other two divisions (the orange division in the league on the left side and the green division on the right side). Finally, at the bottom level, we see that each of the remaining two divisions are split.

The dendrogram illustrates how divisionality forges deeper into the underlying community structure than does modularity. Further, the dendrogram reveals that it is possible to recover higher level community structure by climbing the tree. At $height = 2$ of the divisionality dendrogram, the MLB leagues are recovered. Divisionality reveals a more complete picture of the hierarchal structure in the MLB network. Not only does divisionality reveal the fine-granular divisions, but it is still possible to identify the leagues (as achieved by optimizing modularity) by tracing the divisionality dendrogram upwards.

4.3.2 National Football League (2005)

The National Football League (NFL) consists of thirty two teams split into two conferences (analogous to the leagues of MLB). Within each conference, teams are split into four divisions. In total the league contains eight divisions, each of size four. The schedule consists of each team playing sixteen games. Six of these games are based on intra-divisional play (37.5 percent of the games), as each team plays its three division rivals twice. Furthermore, each division is pitted against one division of the other conference. For these inter-conference divisional matchups, each team of one division plays each team of the other division a single time (for a total of four games per team). The final six games (37.5 percent) occur within the conference. Four of these games result from an intra-conference divisional matchup. The final two games for each team are assigned to create parity in the league. Strong teams from the previous season are matched with other strong teams from the same conference. Weak teams are matched with other weak teams of the same conference.

Modularity splits the NFL network into four partitions. Each of these partitions contains the union of two divisions as dictated by either the inter-conference or intra-conference divisional matchups – the pair of divisions is merged into a single community. With reference to table 4.1, $Q_{opt} > Q_{gov}$ for the NFL, thus confirming that modularity does not favor the official divisional alignments as the optimal community structure for the network.

In contrast, divisionality partitions the NFL network into eight communities. Each community represents one of the eight NFL divisions. Confirming this result, $D_{opt} = D_{gov}$ in table 4.1. This equality establishes that optimization of divisionality favors the official NFL divisional alignments as best describing the community structure based on team schedules.

4.3.3 National Basketball Association (2006)

The National Basketball Association (NBA) has thirty teams split into two conferences. Within each conference, teams are split into three divisions consisting of five teams apiece. Each team plays 82 games in a season. Unlike the MLB and NFL leagues, each NBA team plays every other team. The network is fully connected. Furthermore, unlike MLB, the weights of the NBA network are relatively balanced. Each team plays their division rivals four times (20 games) while playing each team of the opposing conference twice (30 games). The remaining thirty two games are allocated to playing teams in the same conference (either three or four times). This schedule composition is represented by approximate percentages in figure 4.2.

With reference to the NBA schedule, both modularity and divisionality bisect the network. Note that $Q_{opt} > Q_{gov}$ and $D_{opt} > D_{gov}$, indicating that according to both measures, the NBA divisional alignments do not represent the optimal community structure of the league. Rather, in both cases, the recovered partitions match that of the conferences.

This result makes intuitive sense. The NBA divisional alignment is in a sense false since a given team is equally likely to play a divisional foe as it is to play a team from another division within the conference. However, despite the fact that all teams play one another, a given team is twice as likely to play a team from its own conference as it is a team from the opposing conference. Both modularity and divisionality exploit this structure and split the network according to the conference alignments.

4.3.4 National Hockey League (2004)

The National Hockey League (NHL) consists of thirty teams split into two conferences. Within each conference, teams are divided into three divisions of five teams apiece. Each team plays 82 games during the season. The network is fully connected, as each team plays every other team. Teams play their division rivals six or seven games (approximately 33 percent of schedule), teams in the same conference but other divisions four times (49 percent), and teams in the opposing conference one or two times (18 percent).

As shown in table 4.1, $Q_{opt} > Q_{gov}$, thus indicating that modularity does not regard the NHL divisional alignments as representing the optimal community structure for the network. Rather, modularity bisects the network according to the conference alignments. In contrast, divisionality splits the network into six partitions. Note from table 4.1 that $D_{opt} = D_{gov}$ for the NHL, thus indicating that the recovered partitions represent those dictated by the NHL divisional alignments.

4.3.5 NCAA Football (2005)

The NCAA college football (CFB) network contains 119 teams split into twelve conferences (or leagues), for an average league size of 10 teams. Five of these conferences are split into two divisions. As a result, the conference/division alignments create seventeen partitions. For the sake of this analysis, I consider each partition – whether it be a division or a conference – to be the equivalent of a conference. As a result, figure 4.2 depicts only two bars for each college sport (including CFB and CBB of section 4.3.6). Games are categorized as either intra-division (games against teams from the same division/conference) or inter-league (games between teams of different divisions/conferences).

Each team plays approximately twelve games with a strong bias for intra-conference games (67 percent on average). The other third of the games are spread amongst non-conference foes. In conferences that are split into divisions, teams play all of the teams within their division and a few of the teams in the opposing division. With regards to schedule composition, all such games are categorized as intra-division.

A final note of interest involves the *Independents*, a group of four teams. These teams are grouped together; however, they are not regarded as a conference. Three of the teams play two of the other *Independents*. Their remaining ten games are played against teams from *other* conferences.

Modularity divides the NCAA football network into nine communities. The smallest of these communities consists of eight teams – those representing the *Big*

East conference. The largest community contains twenty one teams, merging the two divisions of the *Big Twelve* conference with eight teams of the *Sun Belt* conference as well as the *Independent* team of *Army*. With reference to table 4.1, $Q_{opt} > Q_{gov}$, confirming that modularity does not regard the CFB divisional/conference alignments to represent the optimal community structure for the network.

Similarly, $D_{opt} > D_{gov}$, indicating that divisionality does not regard the divisional/conference alignments as optimal, either. However, divisionality splits the NCAA football network into nineteen communities, two more than the seventeen represented by official alignments. Fifteen of the seventeen actual alignments are recovered exactly as defined by the NCAA. The remaining four divisionality communities result from bisections of the *Pac Ten* and *Big Ten* conferences. Further examination of the schedules of these conferences reveals the reason. In both cases, teams of the conferences are not fully connected. In the *Pac Ten*, each team plays every other team except for one. The *Pac Ten* bisection reflects this imbalance, as each team in a given partition plays the four other teams in its partition but it does not play one of the teams of the other partition. With regards to the *Big Ten* conference (for which the name has remained unchanged for historical reasons despite including eleven teams), each team plays eight of the other ten teams. With the exception of *Wisconsin*, each team within a given partition plays all of the teams within its partition but does not play two of the teams from the opposing partition. *Wisconsin* does not play one team from each partition. In essence, the *Big Ten* is split into *virtual* divisions and divisionality identifies this asymmetry.

4.3.6 NCAA Basketball (2006)

The NCAA college basketball (CBB) network contains 324 teams split into 32 conferences (an average of ten teams per conference). Four of the conferences are bisected into divisions, thus yielding 36 partitions by official alignment. The partition sizes range from 5 to 16 teams. Similar to the the CFB network, there exists one conference deemed the *Independents*, which contains ten teams in this case. Each team plays approximately thirty games. On average, half of these games are played against conference rivals. The exception to this rule involves the *Independents*. Teams within this conference play one another anywhere from two to ten times.

Modularity splits the CBB network into 14 communities (see table 4.1). Six of these communities represent actual conference alignments. The other partitions are the result of merged conferences. The largest such community contains 57 teams and is the result of combining 5 separate conferences. Interestingly, when conferences are merged, they are generally from the same geographic region. The reason for this geographic correlation is that teams play a greater number of their out-of-conference games against teams from the same geographic region. Many of these games are based on in-state or border-state rivalries. Others are based on old conference alignments in which teams were formerly conference foes.

In contrast, divisionality splits the CBB network into 51 communities. Twenty four of the thirty six actual alignments are recovered as they exist. One other community (the *Big West* conference) is merged with *UC Davis* of the *Independents*. This team played half of its games against the *Big West* and only two versus the other *Independents*. Of the remaining eleven conferences, seven of them are bisected. The final four conferences are split into three partitions. Examination of these conferences revealed an imbalance in the intra-conference schedules: certain teams played certain conference foes twice while playing others only once. Divisionality identified these imbalances and split the conferences into smaller groups accordingly.

4.4 Concluding remarks

In this chapter, I introduced a fine-granularity measure of community structure called divisionality. I presented the mathematics of divisionality and then examine its efficacy, in comparison to modularity, by optimizing it on a series of networks with known structure: the schedule structure of six different sports leagues. I demonstrated that divisionality identifies the known, low-level structure of these networks. Further, I showed that the community structure recovered by optimization of divisionality was of equivalent or finer-granularity than that detected by optimization of modularity.

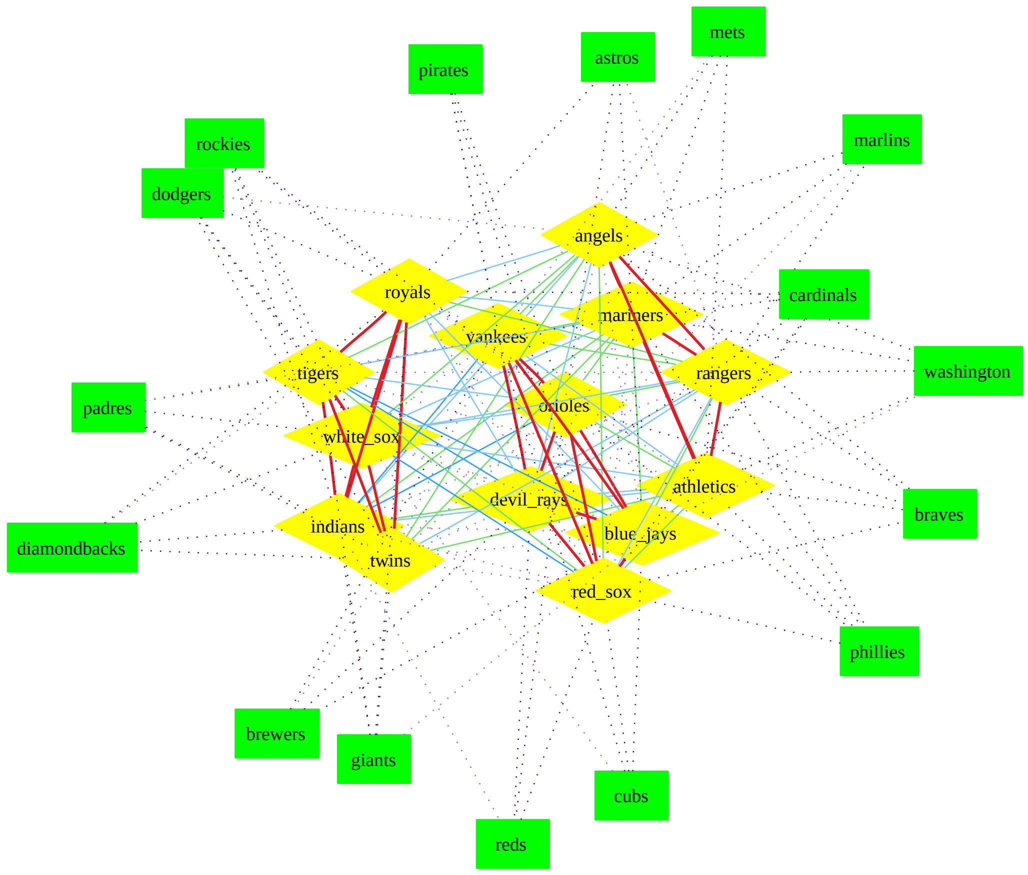


Figure 4.1: Community structure for the 2005 Major League Baseball schedule network based on modularity optimization. The network is split into two communities. The internal community (yellow nodes) represents the American League. The outer community (green nodes) represents the National league. To aid visualization, only edges with at least one end connected to an American League team are shown.

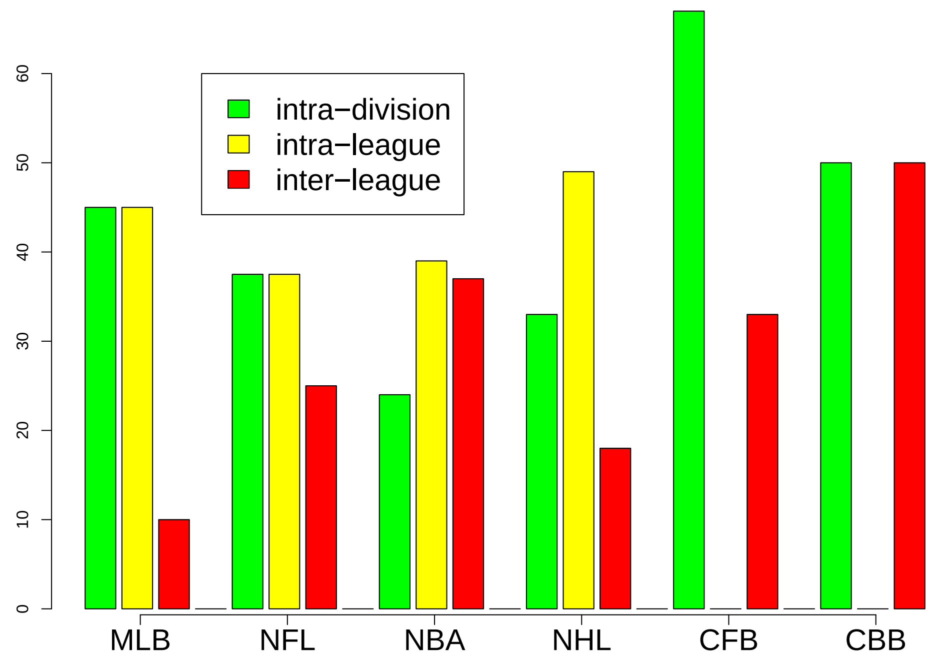


Figure 4.2: Schedule composition for the different leagues.

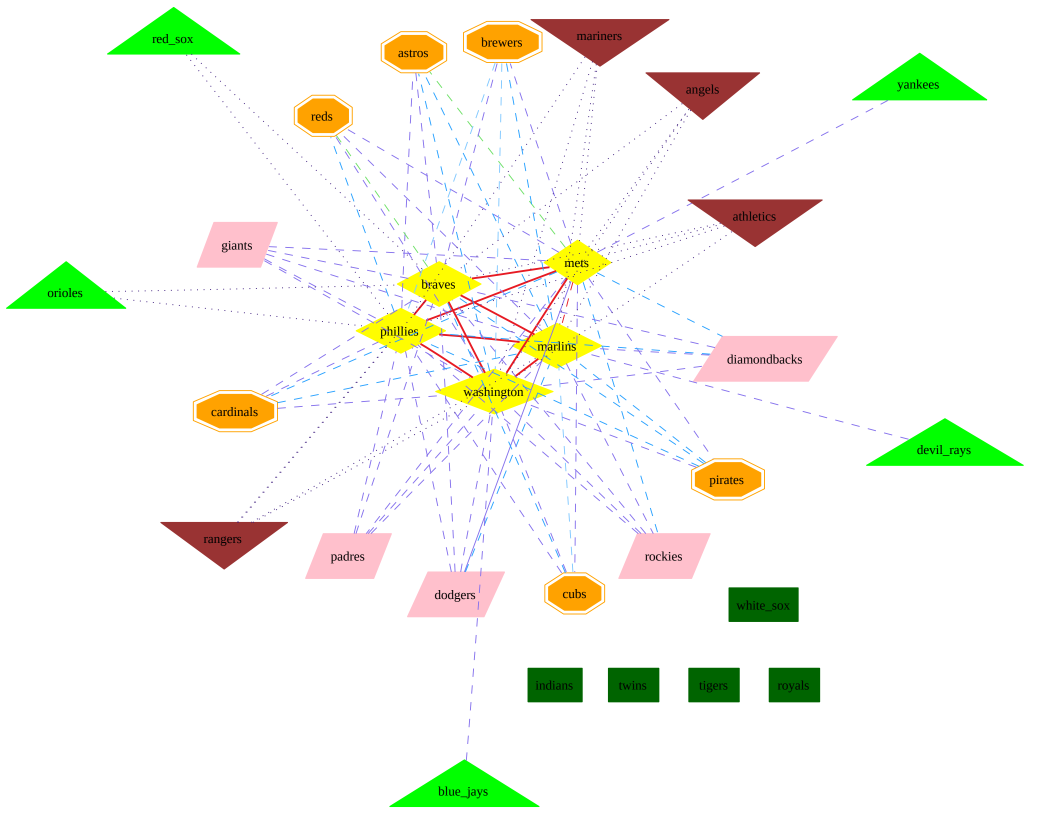


Figure 4.3: Community structure for the 2005 Major League Baseball (MLB) schedule network based on divisionality optimization. The network is split into six communities. Each community represents one of the official MLB divisions. To aid visualization, only edges with at least one end connected to a National League East (NLE) team are shown. The red edges represent those of highest weight and all fall between division rivals of the NLE.

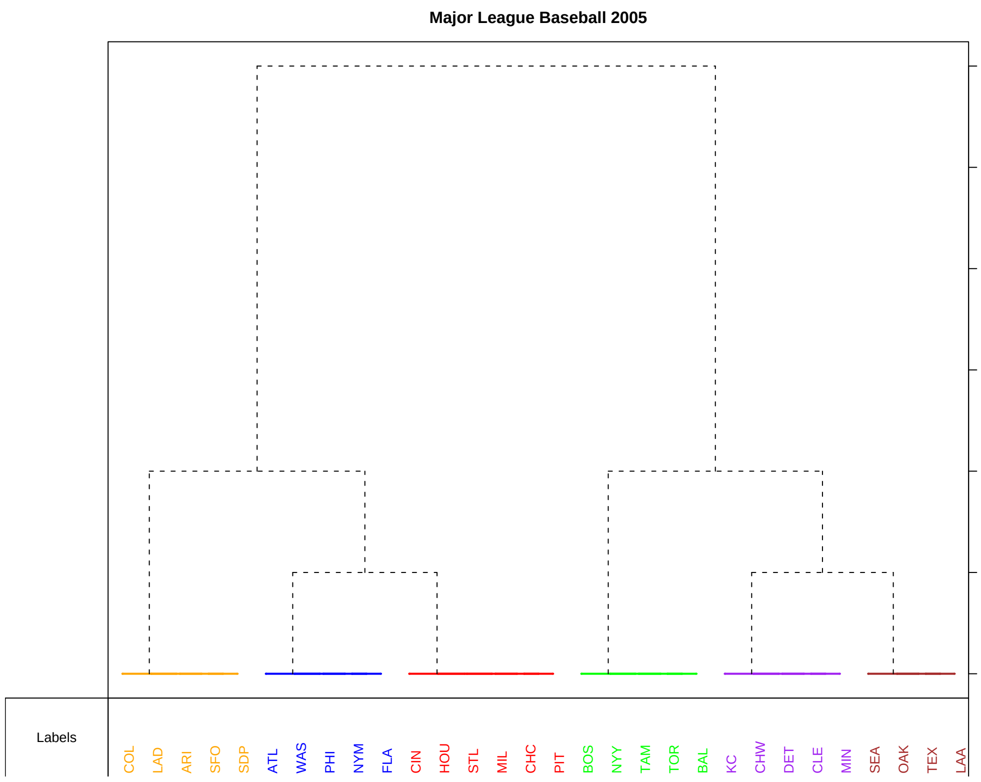


Figure 4.4: This dendrogram of the MLB 2005 schedule network depicts the splits resulting by optimizing divisionality. At the top level (at height 6), all of the nodes are contained in a single partition. At the bottom (height 0), the nodes are partitioned into six communities, each representing an MLB division. At $height = 2$, the two partitions represent the leagues. Modularity failed to find community structure beneath this level.

Chapter 5

A dual assortative measure of community structure

5.1 Introduction

Current community detection algorithms operate by optimizing *modularity*, which analyzes the distribution of positively weighted edges in a network. Modularity does not account for negatively weighted edges. This chapter introduces a dual assortative modularity measure (DAMM) that incorporates both positively and negatively weighted edges. First, I describe the the DAMM statistic and illustrate its utility in a community detection algorithm. Then, I evaluate the efficacy of the algorithm on both computer generated and real-world networks, showing that DAMM broadens the domain of networks that can be analyzed by community detection algorithms.

A *friendship network* is a common example of a complex network. Nodes of the friendship network represent people, and the edges, which are positively weighted, represent friendships. Intuitively, communities are comprised of sub-graphs in the network that are densely connected to one another but sparsely connected to the outside. The term *assortativity* [35] [36] refers to the tendency for nodes to be connected to others that are like, or unlike, them. In the case of the friendship network, communities are based on *positive assortativity* because nodes are connected to others with whom they share a positive connection (friendship). Panel A of figure 5.1 depicts a friendship network, where the solid edges are positively weighted and represent friendships.

In this chapter, I incorporate the concept of *negative assortativity*, or disassortativity, into the definition of community. Nodes are negatively assortative if their connection is based on dissimilarity, rather than likeness. With regards to the friendship network, negatively weighted edges represent the strength of adversarial relationships. I refer to a network that contains only negatively weighted edges as an *adversarial network*. As previously described, all of the edges in the network shown in panel A of figure 5.1 are based on friendships. However, let us assume that this friendship information is unavailable and that instead a list of adversarial relationships between nodes is provided. Further, assume that all pairings in the original network that did not share friendships are now considered adversaries. The resulting adversarial network is presented in panel B of figure 5.1. The dashed edges indicate negative weights. The two networks, the friendship network (top left) and the adversarial network (top right), provide similar but different information. It is not the case that the adversarial network is always the reciprocal of the friendship network.

In this work, I combine the two concepts – both positive and negative assortativity – to form a single definition of community. Because this definition incorporates the contributions of both negative and positive relationships, I refer to it as *dual assortative*. The networks on the bottom of figure 5.1 illustrate this duality. They contain both positive relationships (solid edges) and negative relationships (dashed edges). Such networks may be fully connected, as is the case of the network in panel C of figure 5.1. However, more commonly, only a fraction of the possible relationships between nodes may be known (panel D of figure 5.1). An example of a dual assortative network is one in which the edge weights are based on a similarity measure, such as correlation, which can assume either a positive or negative value. Consider a network where the nodes represent financial traders and the edge weights indicate the correlation of trading behavior between a pair of traders. The dual assortative modularity measure (DAMM) definition incorporates all available information, positive and negative, to assess the strength of community structure.

Intuitively, there exists an asymmetry in the information provided by positive and negative edges. A friendship between two people conveys a stronger bond than sharing a common adversary. However, this is only true when there are three or more communities. When only two *real* communities exist, a negative edge provides the same amount of information as a positive edge. Consider the case of having two communities, community C_a and community C_b. If two nodes share a negative edge, it indicates that one node should belong to C_a and the other to C_b. However, in the case of three or more communities, the negative edge simply indicates that the nodes should reside in separate communities but does not indicate which particular communities the nodes should belong. The information provided by a positive edge is more specific than that for a negative edge.

The remainder of the chapter is organized as follows. In section 5.2, the mathematical framework for the dual assortative measure is introduced and explained. Further, a community detection algorithm used for optimizing the DAMM is described. In section 5.3, I assess the efficacy of optimizing the DAMM on both computer generated and real networks, and section 5.4 summarizes and concludes.

5.2 Methods

This section introduces the mathematics of the dual assortative modularity measure (DAMM), describes the extremal optimization algorithm for optimizing the DAMM on a given network, and describes a measure, called communal overlap, which I use to quantify the fidelity of a community detection result given that the real communities are known and available for comparison.

5.2.1 The dual assortative modularity measure

Before describing the dual assortative modularity measure (DAMM), we review the original modularity measure, which provides the foundation for the DAMM. I then show how to quantify the negative assortative contributions of a network. The positive and negative components are then combined to establish the DAMM. Finally, I introduce the algorithm used to optimize the DAMM on a network.

Positive assortativity

I denote an edge weight between node r and node s as e_{rs}. For simplicity of explanation, I consider only networks with edges of weight $e_{rs} \in \{-1, 0, 1\}$, with $e_{rs} = 0$ implying that the edge is not present. However, in practice e_{rs} will often be a real number, $e_{rs} \in \Re$. Given a set of communities, denoted C, $\{w_{ij} = \sum_{rs} e_{rs} | e_{rs} > 0, r \in C_i, s \in C_j\}$ denotes the cumulative positive edge weight between community i and community j. Similarly, $\{\bar{w_{ij}} = \sum_{rs} e_{rs} | e_{rs} < 0, r \in C_i, s \in C_j\}$ denotes the cumulative negative edge weight between community i and community j.

Equation 5.1 provides the original modularity measure [34], denoted as Q^+. The implied edge weight domain is $e_{rs} \in \{0, 1\}$. The w_{ii} term represents twice the number of positive edges in which both ends terminate at nodes belonging to community i.

Further $a_i = \sum_j w_{ij}$ gives the sum of all positive edge weights with at least one end attached to a node residing in community i, and $T^+ = \sum_i \sum_j w_{ij}$ is twice the total number of positive edges in the network. I use the term *spoke* to refer to the terminal end of an edge. Using this terminology, a_i represents the number of positively weighted spokes connected to community C_i. Similarly, T^+ represents the total number of positively weighted spokes in the network.

$$Q^+ = \sum_{i=0}^{|C|-1} \frac{w_{ii}}{T^+} - \left(\frac{a_i}{T^+}\right)^2 \tag{5.1}$$

The summation in equation 5.1 iterates through the set of communities. For each community, the difference $(w_{ii}/T^+) - (a_i/T^+)^2$ reflects the strength of that particular community. The first term, (w_{ii}/T^+), represents the ratio of intra-communal edges to the total number of edges in the network. An edge is considered to be intra-communal if both ends are connected to nodes residing in the same community. One could mistakenly assume that the higher this ratio, the greater the strength of the community. However, if it were, the ratio would be optimized by a single community containing all nodes within the network. Thus, I compare the ratio found in the first term to the expectation of its value, $(a_i/T^+)^2$. The ratio a_i/T^+ represents the ratio of positive edge spokes connected to the given community to the total number of positive edge spokes, and thus its square gives the expectation. If the difference of terms is positive, the observed ratio is greater than what would be expected if the edges were placed randomly. The greater the (positive) difference, the greater the communal strength. If the difference is negative, the communal strength is found to be weaker than the expectation, suggesting no communal structure for the community being investigated.

The DAMM is given in equation 5.5. It uses a modified form of equation 5.1

to assess the contribution of positively weighted edges. Specifically, I redefine T^+ to be $T = \sum_i \sum_j w_{ij} + |\bar{w_{ij}}|$, so that both positively and negatively weighted edges contribute to the total weight T. The original modularity measure does not account for negative edge weights and thus T^+ does not include the $|\bar{w_{ij}}|$ term.

Negative assortativity

This measure is motivated by the idea that a shared adversary represents a commonality. In other words, if both Jack and Jill are both adversaries with Alice, they share a commonality regardless of whether the pair are friends. In the case of positive weights, the difference between the ratio of edges encapsulated by a community and its expected value (given random edge assignments) was quantified. Here, the scenario is reversed. Adversarial relationships within a community are not desirable. In a scenario of perfect community structure, all negative edges would occur between communities.

Equation 5.2 defines the negative assortative component of the DAMM. The equation resembles that of equation 5.1, except the order of terms is reversed and we consider negatively weighted edges. Here, $\bar{a_i}$ represents the cumulative negative edge weight connected to nodes of community i. In other words, $\bar{a_i} = \sum_j \bar{w_{ij}}$ and $\bar{w_{ii}}$ represents the cumulative negative edge weight encapsulated by community i. The first term of equation 5.2 provides a null test.

$$Q^- = \sum_{i=0}^{|C|-1} \left(\frac{\bar{a_i}}{T}\right)^2 - \frac{\bar{w_{ii}}}{T} \tag{5.2}$$

Dual assortative modularity measure (DAMM)

To establish the DAMM, I combine the negative assortativity contribution of equation 5.2 with the positive assortativity contribution of equation 5.1 (with T^+ replaced by T). I define DAMM as follows:

$$\begin{aligned}
Q^D &= Q^+ + Q^- && (5.3)\\
&= \sum_{i=0}^{|C|-1}\left[\frac{w_{ii}}{T} - \left(\frac{a_i}{T}\right)^2\right] + \sum_{i=0}^{|C|-1}\left[\left(\frac{\bar{a_i}}{T}\right)^2 - \frac{\bar{w_{ii}}}{T}\right] && (5.4)\\
&= \sum_{i=0}^{|C|-1}\left(\frac{w_{ii} - \bar{w_{ii}}}{T}\right) + \left(\frac{\bar{a_i}^2 - a_i^2}{T^2}\right) && (5.5)
\end{aligned}$$

In the absence of negative edges, $T = T^+$, $\bar{w_{ii}} = 0$, and $\bar{a_i} = 0$ for all i, and the DAMM reduces to equation 5.1.

5.2.2 Optimizing DAMM with the extremal optimization algorithm

I use extremal optimization (EO) [7] [6] to optimize the DAMM statistic and detect communities in a network. EO is known to be effective for community detection using the original modularity measure that is based solely on positive assortativity [14]. EO is a divisive approach, in which all nodes are initially placed in a single community. Thereafter, each community is divided recursively into two independent communities, not necessarily of the same size. At each step, the division found to provide the largest increase in modularity is applied, given that the increase is positive. If the best division does not increase modularity, the community is declared *indivisible*. When all existing communities are found to be indivisible, the algorithm halts.

Each division proceeds as follows. Initially, the nodes are randomly assigned to one of two partitions. After all nodes have been assigned, the DAMM is computed. Thereafter, a single node is migrated from one partition to the other, and the DAMM is recomputed by adding ΔQ^D associated with the migrated node (section 5.2.2).

A counter, denoted K, tracks the number of moves since the last DAMM improvement. If the DAMM fails to improve, the counter is incremented. Otherwise, the counter is reset to zero, and the partitioning is recorded along with its associated DAMM value. This partitioning represents the best detected configuration. The process continues until the counter reaches a predetermined threshold. For each division, the size of the community, denoted $|C_i|$, determines the stopping criterion such that the maximum allowable number of steps is $S = \alpha|C_i|$ ($\alpha = 3$ in the experiments of section 5.3). Once the counter reaches the threshold S, such that $K = S$, the process terminates and the best detected configuration is retained. If the split has improved the DAMM value, the global set of communities is updated to reflect the division. Otherwise, it remains unchanged and is marked indivisible.

Calculating ΔQ^D

An important component of the EO algorithm involves choosing which nodes to migrate. Rather than choosing nodes at random, a value $\Delta Q^{D,u}$ is associated with each particular node u. This approach differs slightly from that of [14], which uses a heuristic to ΔQ rather than the exact difference. The value $\Delta Q^{D,u}$ represents the change in DAMM that occurs by migrating the specified node. This method resembles hill-climbing used in other settings and biases the search for an optimal division towards immediate improvements.

In practice, I maintain a list that associates a $\Delta Q^{D,u}$ value with each node u. To select a node for migration, the list of $\Delta Q^{D,u}$ values is ranked and then a node is probabilistically chosen using a method known as τ-EO [14] [7]. Using this process, a node of rank q is chosen with probability of $P(q) \approx q^{-\tau}$ where $\tau = 1 + \frac{1}{\log |C_i|}$. Following the migration of a node u, $\Delta Q^{D,u}$ is updated as described in section 5.2.2.

The calculation of $\Delta Q^{D,u}$ is given by equation 5.7. The derivation is provided in appendix B.1.

$$\begin{aligned}
\Delta Q_t^{D,u} &= \Delta Q_t^{+,u} + \Delta Q_t^{-,u} \qquad (5.6)\\
&= 2\left[\left(\frac{w_{gu} - w_{lu}}{T}\right) + \frac{d_u}{T^2}\left(a_l - a_g - d_u\right)\right)\right] \\
&\quad +2\left[\left(\frac{\bar{w}_{lu} - \bar{w}_{gu}}{T}\right) + \frac{\bar{d}_u}{T^2}\left(\bar{a}_g - \bar{a}_l + \bar{d}_u\right)\right)\right] \qquad (5.7)
\end{aligned}$$

Calculating $\Delta^2 Q^D$

After each migration, all $\Delta Q^{D,u}$ values are subject to change. Rather than recompute the value for each node, a less computationally expensive approach makes use of $\Delta^2 Q^{D,um}$. Each value can be updated as $\Delta Q_{t+1}^{D,u} = \Delta Q_t^{D,u} + \Delta^2 Q_t^{D,um}$, where $\Delta^2 Q_t^{D,um}$ represents represents the change in $\Delta Q^{D,u}$ for node u following the migration of node m at time t.

As noted, computation of $\Delta^2 Q_t^{D,um}$ involves two nodes: the node m that was migrated and the node u for which $\Delta Q^{D,u}$ must be updated. Both nodes might move to the same community, or they might move in opposite directions (each community gains one node and loses the other node). A direction indicator $D \in \{-1, 1\}$ is used to indicate how the nodes move. If they both move to the same community, $D = 1$; otherwise, $D = -1$.

The calculation of $\Delta^2 Q_t^{D,um}$ is given by equation 5.8. The derivation is provided in appendix B.2.

$$\Delta^2 Q_t^{D,um} \quad = \quad 4D\frac{(\bar{d}_u\bar{d}_m - d_u d_m)}{T^2} + \frac{(w_{fs} + \bar{w}_{fs})}{T} \tag{5.8}$$

The computational cost of computing $\Delta^2 Q^{D,um}$ is less than that for $\Delta Q^{D,u}$. The latter requires computing w_{gu}, w_{lu}, $\bar{w_{gu}}$, and $\bar{w_{lu}}$, which is $O(|C_i|)$, where $|C_i|$ represents the number of nodes in community i. The cost of computing $\Delta^2 Q^{D,um}$ is reduced to $O(1)$.

5.2.3 Measuring communal overlap

In section 5.3, I will optimize the DAMM on a given network, recover the detected communities, C^d, and assess the similarity of C^d to the known communities C^k. For this final step, which compares two communities, I introduce a measure that I call *communal overlap*. The statistic quantifies the similarity between two sets of communities and is used to assess the success of each experiment.

The foundation for communal overlap is the Jaccard index, $J(A, B) = |A \bigcap B| / |A \bigcup B|$, which measures similarity between sets, say A and B. Each community, $C_i \in C$, is a set of nodes. Thus, the Jaccard index provides a means for comparing two different community configurations. Let $C^k(n)$ and $C^d(n)$ represent the known and detected communities corresponding to node n. Then, if $A = C^k(n)$ and $B = C^d(n)$ the Jaccard index measures their similarity. In this context, greater similarity implies better detection. Communal overlap, shown in equation 5.9, computes the weighted average of the Jaccard indices for all nodes in the network. The higher the value of communal overlap, $\Omega \in (0, 1]$, the greater the similarity between the communal configurations C^k and C^d, where $\Omega = 1$ represents a perfect match. $\Omega = 0$ is unattainable because at the very least, for all n, $C^k(n)$ and $C^d(n)$ share the node n and thus $|C^k(n) \bigcap C^d(n)| > 0$.

$$\Omega = \frac{1}{N} \sum_{n \in N} \frac{|C^k(n) \bigcap C^d(n)|}{|C^k(n) \bigcup C^d(n)|} \tag{5.9}$$

5.3 Experimental Results

In this section, I report on three experiments. The first two study stochastically generated networks with a prescribed community structure. The third experiment involves a real world network, the 2005 National Football League (NFL) schedule. In each case, I measure the efficacy of the DAMM-enabled EO algorithm to recover known community structure.

5.3.1 Experiment I: independent contributions of positive and negative edges

Generating networks stochastically

In these tests, I generated networks with $N = 64$ nodes and $|C| = 4$ communities of equal size (16). Once the communities are established, both positive and negative edges are added to the network. By default, positive edges are added between nodes of the same community and negative edges are added between nodes of different communities. An exception to this rule involves *false positives* and *false negatives*, discussed in section 5.3.2.

The stochastic generation algorithm uses two parameters: the mean number of intra-community edges per node z_{in} (both nodes in the same community), and the mean number of inter-community edges per node z_{out} (nodes in different communities). Intra-community edges are assigned an edge weight of $e_{ij} = 1$, and inter-community edges are negatively weighted ($e_{ij} = -1$).

For any node $n \in N$ there are $|N| - 1$ possible edges, discounting self-loops. To generate the positively weighted edges, a pseudo-random number $r_{in} \in [0, 1)$ is generated for each potential intra-community edge. If $r_{in} < p_{in}$ the edge is added, where p_{in} is the probability of an intra-community edge existing: $p_{in} = z_{in}/ (|C_i| - 1)$. A similar procedure is followed for negative edges. Each potential inter-community edge is generated with probability $p_{out} = z_{out}/ (|N| - |C_i|)$.

Experimental setup

I refer to the mean cumulative degree of a node, which combines both the intra-community and inter-community degrees, as $z_{cum} = z_{in} + z_{out}$. For the first experiment, I generated a series of networks with $z_{cum} = 16$. While the value of z_{cum} was held static, $z_{in} \in [0, 16]$ and $z_{out} \in [0, 16]$ were dynamically adjusted and relate inversely such that $z_{out} = 16 - z_{in}$. For each (z_{in}, z_{out}) parameter setting, 100 independent networks were generated. I refer to these networks as being dual assortative (DA). The goal is to compare the independent contributions of the positive and negative edges of the DA networks. Towards this end, from each DA network, I extracted the embedded positive assortative (PA) and negative assortative (NA) networks. To extract the PA network, all negative edges were removed from the original DA network. In contrast, to establish the NA network, all positive edges were removed from the DA network. For each network – whether it be a DA, PA, or NA network – the DAMM was optimized using the EO algorithm and the community overlap Ω was assessed.

Any value of $z_{in} \geq |C_i|$ yields full intra-component connectivity. Thus, for $z_{in} = 15$ and $z_{in} = 16$, the intra-component sub-networks are fully connected. On the other hand, the maximum value of $z_{out} = 16$ covers merely one-third of the potential inter-component edge space.

Results

Figure 5.2 shows the results of the first experiment. On the lower x-axis, the positive degree, z_{in}, is displayed. On the top x-axis, the negative degree, z_{out}, is shown. The y-axis represents the mean communal overlap, $\langle\Omega\rangle = 1/|G| \sum_{i=0}^{|G|} \Omega(G_i)$, for the set of generated networks G corresponding to the specified (z_{in}, z_{out}) setting. The solid curve shows results for the DA networks. For all (z_{in}, z_{out}) settings, $\langle\Omega_{DA}\rangle > 0.95$. Optimization of the DAMM on the DA networks detects the known communities with high fidelity. The dashed curve shows DAMM-optimized PA networks. For $z_{in} \geq 4$, the PA networks yield a $\langle\Omega_{PA}\rangle > 0.95$, which is comparable to the DA networks. However, for $z_{in} < 4$, the community overlap values for the PA networks are significantly less than those observed for the DA networks. This deficiency highlights the importance of the negative edges that are removed to create the PA networks. By removing these edges, information used by the DAMM is lost. As a result, the detection process suffers. The dotted curve presents the results for the NA networks. Note that for only $z_{out} = 16$ does $\langle\Omega_{NA}\rangle > 0.95$. For $z_{out} \leq 10$, $\langle\Omega_{NA}\rangle < 0.5$, which means that, on average, there exists less than a 50% overlap between the detected and known communities.

The distance between the NA (dotted) and DA (solid) curves of figure 5.2 highlights the importance of the positive edges that were removed from the original DA networks. The mean distance between the DA (solid) and PA (dashed) curves is 0.084 units of community overlap. By comparison, the mean distance between the DA and NA curves is 0.50 units of community overlap. Removal of the positive edges from the DA networks for the investigated parameter range has a significantly greater deleterious impact on community detection.

These results provide a proof-of-principle for the DAMM. Regardless of the (z_{in}, z_{out}) setting, optimization of the DAMM yields a high communal overlap ($\langle \Omega_{DA} \rangle > 0.95$) on the DA networks. When either the positive or negative edges are removed, the detection process suffers. Note that if the original modularity measure is optimized on the DA networks, the contributions of the negative edges are ignored. By optimizing the DAMM on the PA networks, I achieve the equivalent – since the negative edges have been removed, the negative information is unavailable to the DAMM. Without the negative edges, the communal overlap yield drops. By incorporating the contributions of both positive and negative edges, the DAMM outperforms the original modularity measure. Furthermore, I have demonstrated high fidelity community detection using only the negative edges. At $(z_{in} = 0, z_{out} = 16)$, optimization of the DAMM yields $\langle \Omega_{NA} \rangle > 0.95$.

5.3.2 Experiment II: the impact of false positives and false negatives

The second experiment introduces "false" edges to the networks: *false positives* and *false negatives.* A false positive is a positively weighted edge that connects nodes in different communities. In the language of friends and adversaries, a false positive indicates a friendship between people of different communities. A false negative occurs when two nodes of the same community are connected by a negatively weighted edge. This occurs when there is an adversarial relationship between two people of the same community. False positives and false negatives are routinely found in real-world networks. Their presence contributes to the challenge of detecting communities.

To assess the impact of the false positives and false negatives, we generated DA networks with a fixed (z_{in}, z_{out}) setting, and then exclusively added either false positives or false negatives. For this experiment, I did not disassemble the DA networks into their PA and NA constituents. The community detection algorithm, which optimizes the DAMM, was applied to each DA network and the communal overlap Ω was computed.

Generating false positives and false negatives

To include false positives and false negatives, I introduce two additional parameters: f^+, the mean number of false positives per node, and f^-, the mean number of false negatives per node. To generate false positives, I assess the unused negative edge space following the initial edge generation phase (section 5.3.1). Assume that E^- represents the entire negative space considered for a given node in the initial phase. I refer to the unused subset of this space as $U^- \in E^-$ and establish the probability $\phi^+ = f^+/U^-$. For each potential edge $e_{ij} \in U^-$, a random number is generated, $r^+ \in [0, 1)$. If $r^+ < \phi^+$, a positive edge between is added such that $e_{ij} = 1$, where i and j are known to reside in different communities. The generation of false negatives follows a similar procedure; however, f^- dictates the likelihood of adding negatively weighted edges between nodes residing in the same community.

Experimental setup

I generated base networks with three different settings: $(z_{in} = 5, z_{out} = 16)$, $(z_{in} = 7, z_{out} = 16)$, and $(z_{in} = 5, z_{out} = 22)$. The first parameter pair was chosen because the degree represents one-third connectivity within both the intra-community and inter-community subspaces. For a given node, since $|C_i| = 16$, the maximum number of intra-component edges is $z_{in} = 15$ and the maximum number of inter-component edges is $z_{out} = 48$. Figure 5.2 shows that $z_{in} = 5$ represents the relative threshold for which $\Omega_{PA} > 0.95$ and $z_{out} = 16$ for $\Omega_{NA} > 0.95$. The other two parameter settings were chosen to analyze the effect of independently increasing intra-community or inter-community connectivity. The second pair of parameters, $(z_{in} = 7, z_{out} = 16)$, was chosen to highlight the effect of increasing z_{in} when z_{out} is maintained. The increase from $z_{in} = 5$ to $z_{in} = 7$ represents a 13% increment in intra-community coverage. Analogously, the third parameter pair, $(z_{in} = 5, z_{out} = 22)$, represents a 13% increase in the inter-community coverage and allows us to analyze the effect of increasing z_{out} while holding z_{in} steady. To these base networks, I independently added either false positives or false negatives. Accordingly, the edge generation parameter space is extended to $(z_{in}, z_{out}, f^+, f^-)$ with $f^+ \in [0, 8]$ and $f^- \in [0, 8]$. None of the networks contain both false positives and false negatives: the addition of the false edges is mutually exclusive to a single type. Thus, if $f^+ > 0$, then $f^- = 0$; conversely, if $f^- > 0$, then $f^+ = 0$. For each parameter setting, forty networks were created, each with a different random number generator seed.

Results

Figure 5.3 shows the results on networks with false positives and false negatives. In the top graph, corresponding to $(z_{in} = 5, z_{out} = 16, f^+, f^-)$, both curves are similar, although the detrimental effect of the false negatives is slightly greater. As expected, as the rate of either false positives or false negatives increases, $\langle\Omega\rangle$ decreases. When only a couple of false edges are added, the known communities are detected without a significant drop-off. For $f^+ < 3$ and $f^- < 3$, the mean communal overlap exceeds $\langle\Omega\rangle = 0.95$. However, beyond this range, the detection rate suffers. With reference to table 5.1, the mean communal overlap for both curves, $M = \langle(\Omega^+ + \Omega^-)/2\rangle$, is 0.67.

In the middle graph, corresponding to $(z_{in} = 7, z_{out} = 16, f^+, f^-)$, and with the mean degree of intra-community edges increased from $z_{in} = 5$ to $z_{in} = 7$, the effect of the additional positive edges is observable. Note that for $f^+ < 6$ and $f^- < 4$, the mean communal overlap $\langle\Omega\rangle > 0.95$. Thus, the range of high fidelity detection has been extended. Comparison to the top graph highlights another effect of the additional information: the mean distance between the false positives curve and the false negatives curve, denoted as $\langle\delta\rangle$, has increased. With reference to table 5.1, $\langle\delta_{z_{in}=5,z_{out}=16}\rangle = .058$ as compared to $\langle\delta_{z_{in}=7,z_{out}=16}\rangle = .113$. Further, the increase in z_{in} improves the mean communal overlap for the range of both curves from $M_{z_{in}=5,z_{out}=16} = .67$ to $M_{z_{in}=7,z_{out}=16} = .82$.

The bottom graph presents results for $(z_{in} = 5, z_{out} = 22, f^+, f^-)$, for which, in comparison to the top graph, z_{out} is increased and z_{in} is unchanged. Similar to the effect observed in the middle graph, M increases in comparison to the initial parameter setting ($M_{z_{in}=5,z_{out}=22} = .79$ as compared to $M_{z_{in}=5,z_{out}=16} = .67$). However, unlike the case for increasing z_{in} (top graph), the mean distance between the curves, $\langle\delta\rangle$, does not differ significantly ($\langle\delta_{z_{in}=5,z_{out}=22} = .069\rangle$ compared to $\langle\delta_{z_{in}=5,z_{out}=16} = .058\rangle$).

The second experiment establishes that the independent impact of false positives and false negatives is influenced by the composition of the network. An increase in either z_{in} or z_{out} lessens the detrimental effect of either false positives or false negatives, as demonstrated by the M values of table 5.1. Further, the additional edges (relating to z_{in} and z_{out}), appear to have an asymmetric effect on the impact of false positives and false negatives. The increase in z_{in} significantly widens the gap between the false positives curve and the false negatives curve ($\langle\delta_{z_{in}=7,z_{out}=16}\rangle = .113$). The increase in z_{out} has a much less pronounced effect on the distance between the false positives and false negatives curves ($\langle\delta_{z_{in}=7,z_{out}=16}\rangle$=.069).

Table 5.1: Results for false positives and negatives.

z_{in}	z_{out}	$\langle\Omega^+\rangle$	$\langle\Omega^-\rangle$	$\langle\delta\rangle$	$M = \langle\frac{\Omega^+ + \Omega^-}{2}\rangle$
5	16	.70	.64	.058	.67
7	16	.88	.77	.113	.82
5	22	.83	.76	.069	.79

5.3.3 Experiment III: 2005 National Football League schedule

The third experiment uses a real dataset: the 2005 National Football League (NFL) schedule. From the dataset, I construct networks representing the correlation of team schedules. Each team is represented by a node. Edges between nodes are weighted to indicate the correlation of the two team's schedules. Teams that play similar opponents show positive correlations. Teams that play dissimilar schedules are negatively correlated. The network contains both positively and negatively weighted edges and is thus dual assortative. Because it is possible to compute the correlation between any two team schedules, the network is fully connected. However, the objective of the experiment is to examine the efficacy of optimizing the DAMM on partial representations of the dual assortative network. Accordingly, only a subset of the possible edges are represented in any given generated network.

The NFL consists of thirty-two teams split into two conferences. Within each conference, teams are grouped into four divisions of four teams apiece. Each team plays sixteen games. Six of these games are the result of a team playing its three division rivals twice each. In addition, each division is paired with one division of the same conference and a second division that resides in the other conference. For each team, these division-versus-division games account for eight additional games (bringing the running tally to fourteen games). The final two opponents for each team result from games against teams from the same conference, but not involved in the division-versus-division matchup. In total, each team faces thirteen unique opponents.

Through extensive analysis using the EO algorithm, I identified four optimal and two near-optimal communal alignments for the fully connected NFL schedule correlation network. I refer to these communal alignments as the *known* optimal configurations. The four optimal alignments each consist of three communities (one community consisting of 8 teams and the other two communities containing 12 teams apiece). In each case, the 8 team community is comprised of two divisions from the same conference that are pitted in a division-versus-division matchup. Each of the 12 team communities contain three divisions, with one division being involved in an intra-conference division-versus-division matchup with one of the other divisions and an inter-conference division-versus-division matchup with the remaining division. Both of the near-optimal communal alignments consist of four communities. In one case, each community contains two divisions pitted in an inter-conference division-versus-division matchup. In the other, each community consists of two divisions pitted in an intra-conference division-versus-division matchup.

With regards to the four optimal communal alignments, the NFL schedule correlation network contains both *false positives* and *false negatives*. In each alignment, there exist teams sharing positively weighted edges that belong to different communities. These edges constitute the false positives. Further, in each alignment, there exist teams within the same community that share negatively weighted edges. These edges constitute false negatives.

Generation of networks

To study the performance of the DAMM on the NFL network, I first optimized it on various *partial* representations of the NFL schedule correlation network. The detected communal alignment was then compared to the set of known optimal configurations and identified the closest match. The best communal overlap score from this series of comparisons was recorded as the communal overlap value.

The fully connected NFL schedule correlation network contains 992 edges (discounting self-loops). Of these edges, 352 (35 percent) are positively weighted and 640 are negatively weighted. We generated the partial representation using a procedure similar to the edge generation algorithm used to generate the partial representations described in section 5.3.1. Instead of computing probability thresholds (such as p_{in} and p_{out}) from a mean degree (such as z_{in} and z_{out}), I simply used the probability thresholds as parameters. Each possible positively weighted edge of the full representation was selected with probability p^+ and each negative edge was selected with probability $p^- = 1 - p^+$.

The stochasticity of this process guarantees that with high probability individual nodes of a partial representation will have varying degree. Because of this asymmetry, certain nodes are more difficult to classify than others. These asymmetries could yield situations where the optimal experimental DAMM configuration will not concur with the known optimal configurations. In such a case, a sub-optimal communal overlap will result.

The objective is to examine whether, on average, the DA information utilized by the DAMM leads to better community detection relative to either the PA or NA information in isolation. Recall that I create the PA network by removing all negative edges from the corresponding DA network; whereas, for the NA network, I remove all positive edges from the DA network.

Experimental setup

I explored the parameter range $p^+ \in [0, 1]$ and $p^- \in [0, 1]$. All of the studied networks were partial representations of the fully connected network. For each (p^+, p^-) setting, 40 networks were generated. Similar to the experiment of section 5.3.1, in each case, I separately optimized the DAMM on the DA network, the PA network, and the NA network.

Results

Figure 5.4 gives the results of the third experiment. The bottom x-axis represents the p^+ values; the top x-axis represents the p^- values. The y-axis represents the mean communal overlap, $\langle \Omega \rangle$, for the corresponding (p^+, p^-) thresholds. Each data point represents the mean communal overlap resulting from optimization of the DAMM on 40 independent, randomly generated partial networks with the same prescribed thresholds.

For each (p^+, p^-) setting, optimization on the DA networks yields an equal or higher $\langle \Omega \rangle$ value than for either the PA or NA networks. Only at $(p^+ = 1, p^- = 0)$ and $(p^+ = 0, p^- = 1)$ do $\langle \Omega_{PA} \rangle = \langle \Omega_{DA} \rangle$ and $\langle \Omega_{NA} \rangle = \langle \Omega_{DA} \rangle$, respectively. At these settings there are either exclusively positive or exclusively negative edges, and thus, these DA networks are equivalent to the respective PA or NA cases. For all parameter settings at which there are both positive and negative edges, the DAMM uses both types of information and achieves higher mean communal overlap values. Despite the presence of 1.8 times more negative edges than positive edges, the positive edges provide more information for community detection. Using only negative edges, at $(p^+ = 0, p_- = 1)$, optimization of the DAMM detects a sub-optimal communal alignment. On the other hand, using only positive edges (at $(p^+ = 1, p_- = 0)$) optimization yields an optimal mean communal overlap value. Figure 5.4 highlights the asymmetry regarding the amount of information provided by the positive edges as compared to the negative edges. The mean distance between the DA and PA curves is 0.20 communal overlap units; whereas, the mean distance between the DA and NA curves is 0.34 communal overlap units. The positive edges contribute more to the community detection process. As expected, as p^+ increases $\langle \Omega_{PA} \rangle$ increases. Similarly, as p^- increases, $\langle \Omega_{NA} \rangle$ increases.

5.4 Summary and conclusions

The DAMM provides a way to assess community structure in networks containing both positively and negatively weighted edges. This extends the paradigm of the friendship network to that of a *friends and adversaries* network. Negative information, previously ignored, now provides useful, additional information to community detection algorithms.

The efficacy of the DAMM was demonstrated, both for stochastically generated synthetic networks and a real-world example based on the 2005 NFL schedule. Furthermore, the experiments revealed the asymmetry in the information provided by positive and negative edges. This asymmetry is due to the greater specificity provided by a positive edge given that more than two communities exist.

The contributions of the DAMM are two-fold. First, it is now possible to analyze networks containing solely negative information. Second, the DAMM improves community detection in networks containing both positive and negative information. An example of such a network is one in which edge weights are based on a similarity metric, such as correlation, that can assume either positive or negative values. The NFL schedule correlation network of section 5.3.3 provides a real-world example. The DAMM expands the domain of problems for which community detection algorithms can be applied.

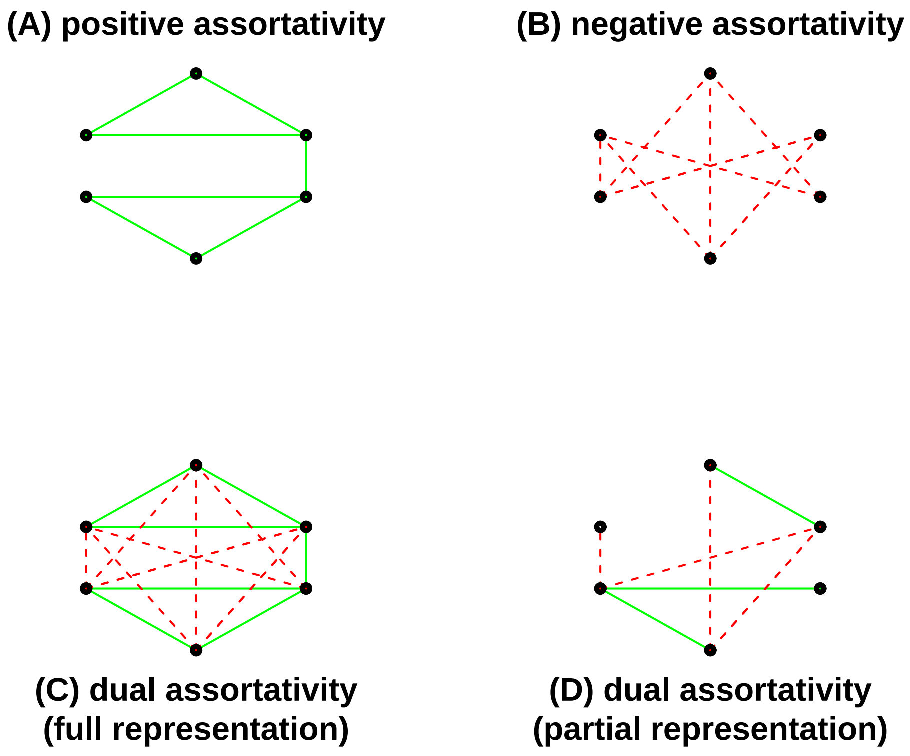

(A) positive assortativity
(B) negative assortativity
(C) dual assortativity (full representation)
(D) dual assortativity (partial representation)

Figure 5.1: Comparison of networks with different types of assortativity. Solid edges denote positive edge weights; dashed edges denote negative edge weights. The network in panel A portrays positive assortativity (PA) and provides an example of a friendship network. In panel B, the network strictly contains negative edges and depicts negative assortativity (NA). I refer to this type of network as an adversarial network. The network in panel C exemplifies a fully connected dual assortativity (DA) network. Here, both positive and negative edges are present. In panel D, a partially connected dual assortative network is illustrated. The dual assortative modularity measure (DAMM) can be used to assess community structure in all of the above cases. The partially connected DA network of the bottom right is the most general, and it is these types of networks that I study in section 5.3.

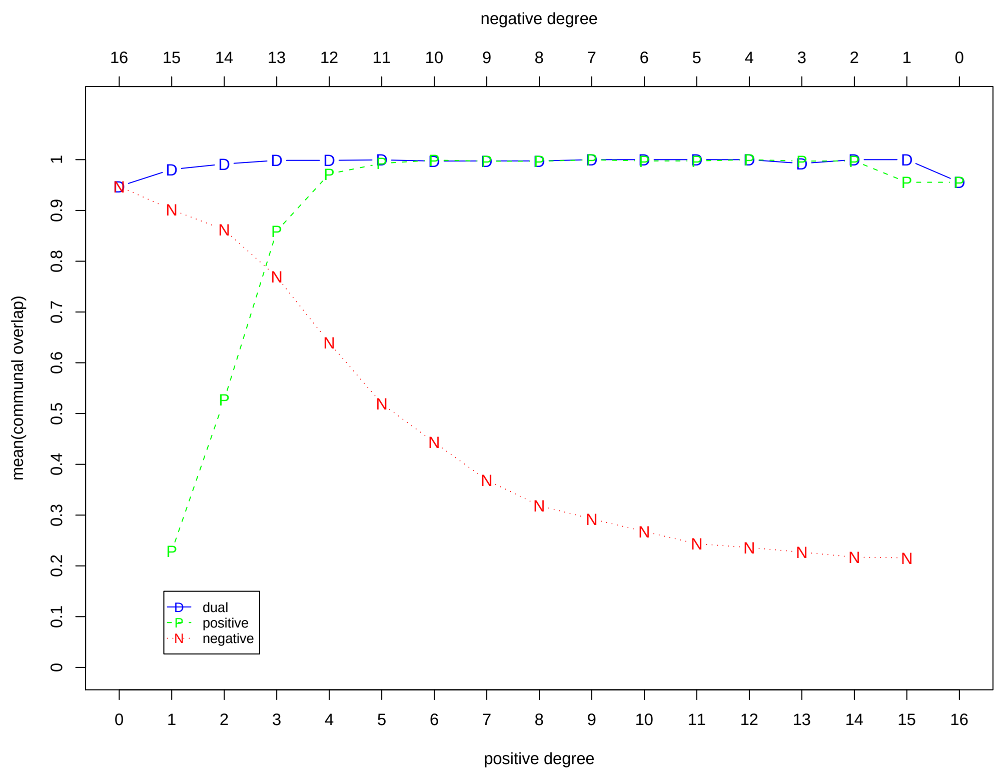


Figure 5.2: Community overlap comparison for dual assortative (solid), positive assortative (dashed), and negative assortative (dotted) networks. Each data point represents the mean communal overlap for optimization of the DAMM on 100 independent, computer generated networks.

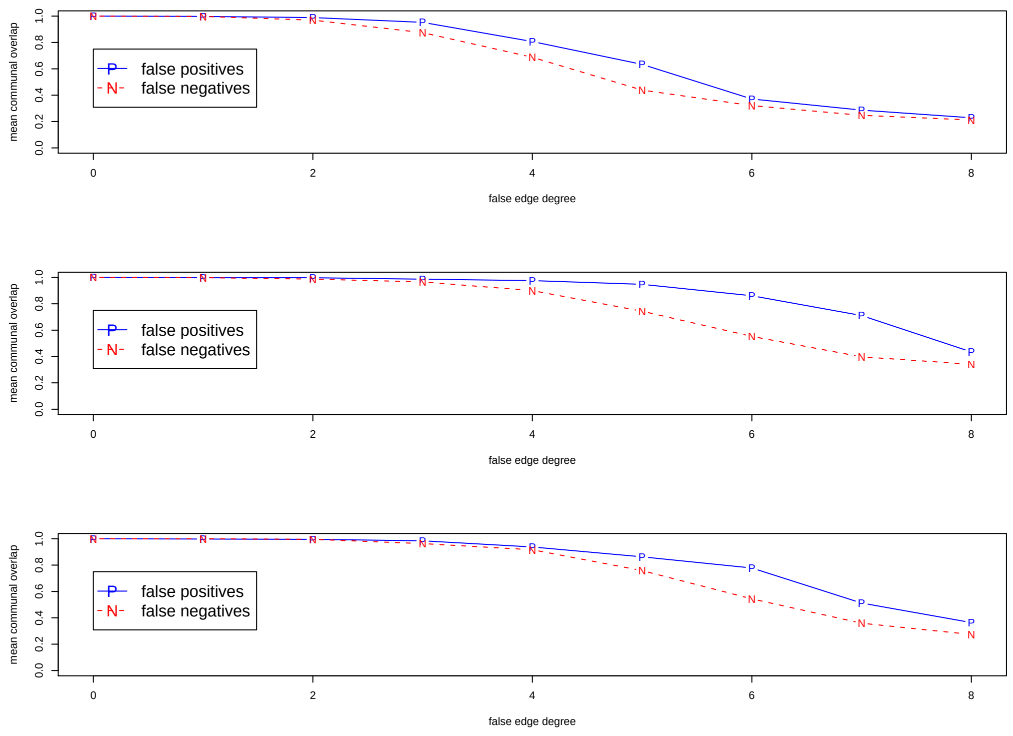


Figure 5.3: The effect of false positives (solid) and false negatives (dashed) on communal overlap. The top graph represents base networks with ($z_{in} = 5, z_{out} = 16$); the middle graph for networks with ($z_{in} = 7, z_{out} = 16$); and the bottom graph for networks with ($z_{in} = 5, z_{out} = 22$). For each graph, the y-axis represents the mean communal overlap value for forty networks. The x-axis represents the mean number of either false positives or false negatives added to the networks.

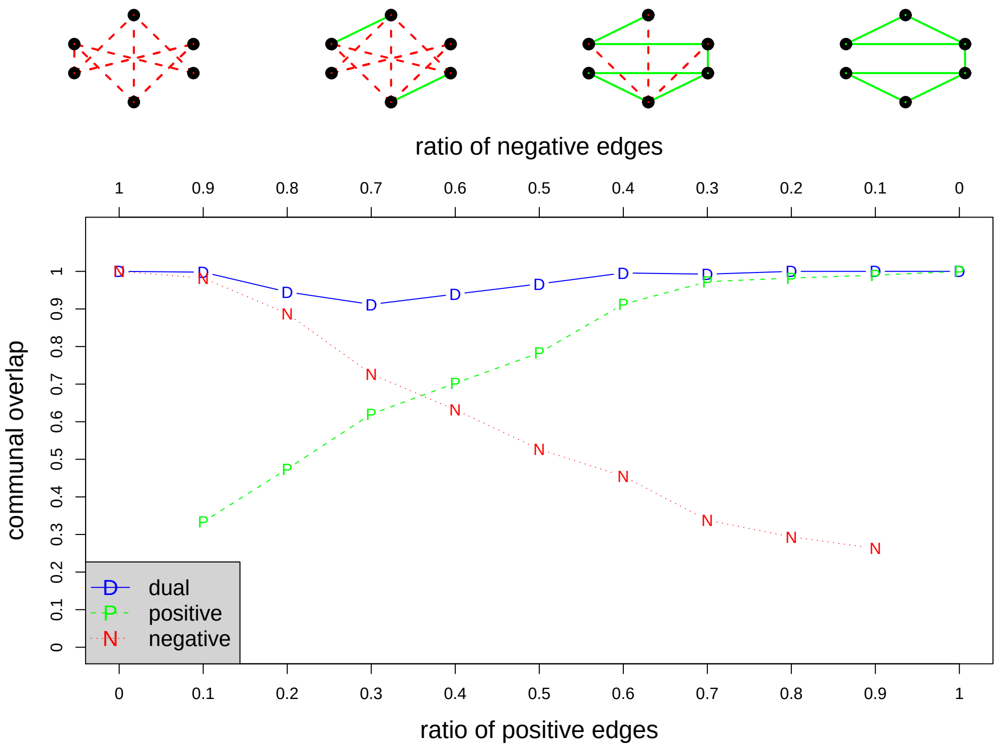


Figure 5.4: Community overlap measures for the 2005 NFL schedule network. The bottom x-axis represents ratio of positive edges present (p^+) and the top x-axis represents the ratio of negative edges present (p^-). The networks above indicate the change in positive/negative edge composition as the graph is traversed left to right (dashed edges represent negative edges and solid edges represent positive edges). The y-axis represents the mean communal overlap, $\langle\Omega\rangle$. The different curves present information regarding the different types of networks upon which the DAMM is optimized: DA networks (solid), PA networks (dashed), and NA networks (dotted). Optimization of the DAMM on the DA networks yields as good or better mean communal overlap values than for either the PA or NA networks. By utilizing both the positively and negatively weighted edges, optimization of the DAMM provides better community detection than the original modularity measure that operates only on positively weighted edges.

Part II

Evolving traders in a simulated financial market

Chapter 6

Evolving traders in an artificial market ecosystem

6.1 Introduction

The research described in this chapter represents the first step of an ambitious goal: to evolve a market ecosystem that generates dynamics similar to those seen in real markets. A simulator of this quality would provide an invaluable tool for the study of financial markets. Before tackling an entire market ecosystem, one must first identify a useful evolutionary framework to handle the task. This chapter describes a stack-based genetic programming approach that evolves a single trader. Towards this objective, I ask the following question: is it possible to evolve a single trader capable of achieving consistent profits in a random market inhabited by *noise traders* [5] [20] [17]?

This chapter demonstrates how genetic programming can evolve programs that exhibit complex trading behaviors. It is organized as follows. Section 6.2 introduces the fundamental components of the simulated market environment. Section 6.3 explains the genetic programming (GP) framework – including an introduction to both the software architecture and the stack-based language developed specifically for the genetic programs. Section 6.4 reports the results of two independent experiments using the GP framework. For each experiment, I present the mechanics of a successful trading strategy. Section 6.5 provides conclusions.

6.2 Defining the market ecosystem

In this section, I introduce the components of a simplified market ecosystem, where a single asset is traded. Section 6.2.1 introduces the market clearing mechanism – a continuous double auction (CDA). With regards to the market ecosystem, the auction provides the environmental infrastructure. The description of the CDA focuses on the data structure used to conduct the auction: the *limit order book*. Section 6.2.2 introduces the organisms that inhabit this simplified ecosystem.

6.2.1 The environmental infrastructure: a continuous double auction

The continuous double auction is the most widely used market clearing mechanism in modern financial markets. The auction is considered *double* because traders can submit both buy and sell orders, and it is considered *continuous* because traders may submit orders at any time. Orders supplied to the CDA are cleared using a limit order book (LOB). Figure 6.1 illustrates a LOB. Price is represented on the x-axis; number of shares are represented on the y-axis.

I consider two types of orders that can be submitted: a limit order and a market order. A *limit order* specifies the following fields: the quantity of shares to be traded, whether to buy or sell, a limit price, and an expiration time. Limit orders do not generate immediate transactions. Rather, they are cached on the LOB. For each unique price, a priority queue exists. Incoming limit orders are placed at the end of the queue associated with the specified limit price. As such, older limit orders are given precedence over newer limit orders. If a limit order resides on the LOB at its expiration time, the order is removed from the associated queue. In addition, limit orders can be cancelled at any time. The *limit price* dictates the transaction price for the order.

Market orders constitute the second type of order. In contrast to limit orders, market orders trigger immediate transactions. Market orders specify only two fields: the quantity of shares to be traded and whether to buy or sell. Significantly, market orders do not designate a transaction price. Rather, the transaction price is dictated by the best available price on the other side of the LOB. Buy market orders eliminate sell limit orders. Sell market orders remove buy limit orders. The number of shares removed is dictated by the size of the order.

The lowest selling price at any time t is referred to as the best ask, $a(t)$. The highest buy price at any time t is called the best bid, $b(t)$. The *midpoint price*, $p(t) = \frac{a(t)+b(t)}{2}$, is used as an approximation of the two values. The gap between the best bid-ask prices, $s(t) = a(t) - b(t)$, is called the bid-ask *spread*. The best prices change in three ways: either by the arrival of new market orders, the cancellation of queued limit orders at the current best prices, or the arrival of limit orders that fall within the current spread region. From this point forward, it is implied that all simulated markets employ a continuous double auction.

6.2.2 The organisms

Three broad approaches exist for constructing the agent-based trader population: agents may be handcrafted, agents may be evolved, or a combination of the two

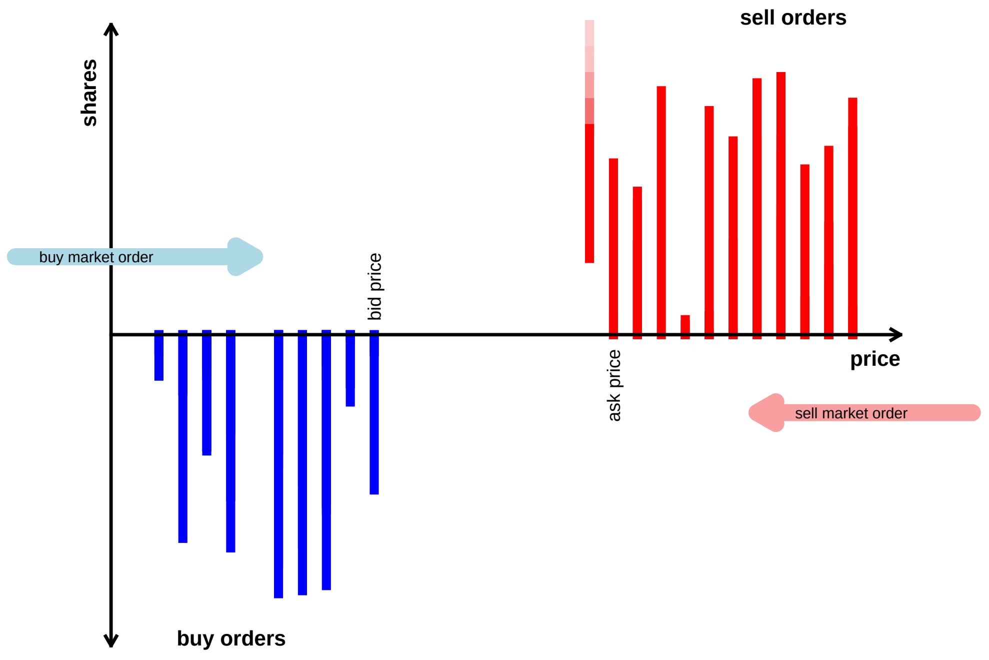


Figure 6.1: Illustration of the limit order book. The vertical bars represent limit orders. Horizontal arrows represent market orders. Blue designates buy orders; red designates sell orders. The arrival of a sell limit order is depicted as falling on to the LOB.

approaches can be utilized. I employ the latter option. The first option, that of exclusively handcrafting agents, resembles a creationist approach and conflicts with the future goal of coevolving a population of traders. However, as influenced by [17], I subscribe to the belief that noise traders fulfill a critical role in the supply and demand of liquidity to the market. Thus, hardwired noise traders are integrated and maintained in all of the simulated market experiments. Section 6.2.2 introduces the mechanics of the noise trader. The objective in this chapter is to evolve a single trader that can produce consistent profits in an environment populated by noise traders. Accordingly, the other inhabitant of the market environment is a single genetic programming agent. Looking ahead, section 6.3.1 provides an explanation of how the GP agent interacts with the market environment.

Noise traders: the base organisms

Noise traders play a critical role in the market ecosystem [17]. They provide a constant flux of orders, or liquidity, to the market. Ecologically, the role of noise traders is similar to that of autotrophs in a grassy plains ecosystem, which trap sunlight and provide a source of energy.

I adopt the noise trader model proposed by [16], which has two types of noise traders. One type places market orders; the other places limit orders. In both cases, the decisions about when to place orders is dictated by a Poisson process. Impatient noise traders place market orders with a Poisson rate of μ shares per unit time. In contrast, patient noise traders place limit orders randomly in both price and time. Buy limit orders are distributed uniformly within a semi-infinite interval $-\infty < p < a(t)$, where p represents price. Similarly, sell limit orders are placed within the semi-infinite interval $b(t) < p < \infty$. Both buy and sell limit orders arrive with the same Poisson rate density, denoted as α. The units of this density are measured as shares per unit price per time, with each price tick represented by a separate price priority queue. All orders are assumed to be of unit size. As explained in section 6.2.1 queued limit orders can be cancelled. The rate of cancellation, δ, is dictated by a Poisson process as well. The rates of buying and selling are maintained to be equal. Accordingly, the decision of whether to buy or sell is dictated by an unbiased coin flip.

For simplicity, in the simulations, a single noise trader of each type is maintained. The α, μ, δ, and λ parameters are calibrated such that they represent the cumulative order rates observed in a real market (the high liquidity Shell stock of the London Stock Exchange). The noise traders are provided unlimited credit and thus continue to trade regardless of profit or loss.

The two types of noise traders guarantee a dynamically changing environment. Limit order noise traders provide orders to be queued; market order noise traders remove the queued orders. The interaction of the two, each dictated by a random process, causes the bid and ask prices to move over time.

6.3 Genetic programming framework

In this section, I introduce the genetic programming (GP) framework. First, a GP architecture for evolving a single profitable trader is defined. Next, I introduce a stack-based programming language (called Staq) designed specifically for genetic programming.

6.3.1 Genetic programming architecture

Before outlining the GP architecture, I reiterate my immediate objective: to evolve a single profitable trader. Accordingly, I adopt a generational, rather than steady-state, GP approach. The first step of the generational approach involves creating an initial population of GP agents with randomly generated programs. Next, the fitness of each agent is assessed by isolating it in a market inhabited by noise traders and observing its performance. Based on these fitness values, a new generation of GP agents is created. The higher the fitness of a given agent, the greater the chance that it will contribute to the future generation. Genetic operators serve both to explore combinations of strong genes resident in the current gene pool as well as to inject new candidate sequences. The new generation replaces the old generation and the process continues until either a prescribed fitness goal or maximum number of generations is reached.

Figure 6.2 illustrates the software architecture for the GP framework. I refer to the software entity that manages the population of GP agents as the *GP engine*. The GP engine is responsible for both dispatching GP agents for fitness assessment and creating subsequent generations.

Fitness assessment of an individual GP agent involves testing its performance in a market inhabited by noise traders. Each GP agent connects to its own market. Within a market environment, three traders exist: a proxy trader, a limit order placing noise trader, and a market order placing noise trader. The proxy trader provides the GP agent a gateway to the market. It serves two purposes. Firstly, the proxy trader submits orders to the market on behalf of the GP agent. Secondly, the proxy trader provides the GP agent with market state information.

To assess the fitness of a GP agent, the market is executed for a prescribed number of timesteps – the approximate equivalent of 10000 market events, where an event is defined as either an order submission, a transaction, or a cancellation. The noise traders of each market are provided identical μ, α, δ, and λ parameters (each explained in section 6.2.2). However, each market is provided a different random number generator seed. By providing similar but not identical market environments, the evolution of robust trading strategies is encouraged.

Before allowing the GP agent to interact with the market, a *saturation* period is required. During this period, the noise traders interact until the volume of orders on the LOB reaches an approximate equilibrium. During the latter stage of the saturation period, the proxy agent collects LOB state information for the GP agent (see section 6.3.2). This information can be used by the GP agent in its decision process. Once activated, the GP agent executes its associated program on each discrete market timestep. During program execution, the GP agent has the opportunity to generate orders. If orders are created, the proxy trader submits them to the LOB on behalf of the GP agent.

6.3.2 Staq: A stack-based GP language

I designed a GP language, called Staq, for these experiments. The GP agents execute Staq programs. Through the application of genetic operators, the Staq programs are evolved. As suggested by the name, the language uses global stacks for passing arguments to instructions. Staq is similar to Push [55], which descended from Forth [48]. The Staq language is implemented in Java and is interpreted.

Staq is multi-typed. Primitive types include: *integer*, *float*, *boolean*, *name*, and *code*. Each type is represented by an independent stack, and instructions are associated with each type. Execution of an instruction involves popping the required arguments from the appropriate stack(s), executing the instruction, and pushing the result to the appropriate stack.

Construction of the Staq language began prior to publication of Spector's seminal work on Push and for software compatibility reasons is still the GP language for this project. An effort was made to align the instruction set of Staq to that of the second version of Push. The stack-based approach lends itself to the task of representing the multi-typed data of a stock market. Furthermore, the ability to create extensible types provides a mechanism for bundling different types to represent objects commonly found in markets. For instance, the *order* type (as explained in section 6.3.2) is used extensively. Traditional GP approaches [26] use tree-like structures to represent programs and employ a single data type. Representation of the *order* type with a traditional approach proves challenging. The advantages provided by extensible typing motivated creation of the Staq language.

Within a Staq program, individual instructions are considered atoms, but they may be grouped together and represented as a list. The term *expression* is used to refer to a single unit that may reside on a stack. An expression may consist of a value, an atomic instruction, or a list of instructions. The size of a Staq program is measured in *points*. Each value, instruction, and list contributes a single point to the total size. Similar to Push, Staq supports both dynamic execution of code sequences and variable binding. By associating a variable name with a code expression, dynamic function creation is possible.

Extensible types: order and market

An additional feature of Staq, similar to Push, is the possibility of *extensible types*. Different types may be bundled together to create new object types. Each extensible type is represented by an independent stack. Upon construction, the required arguments are popped from their associated stacks, the extensible type is created, and the object is pushed to its stack. The availability of extensible types allows the language to provide high level functionality for tailored tasks. This feature was used for the task of evolving market agents. Two extensible types were created: the *order* type and the *market* type. These types provide Staq agents a portal to the simulated market environment.

The *order* type allows for the creation of orders using constructors for either market or limit orders. During GP execution, Staq agents submit orders to the simulated LOB by constructing objects of the *order* type and leaving them on the associated stack. After a GP agent finishes executing its Staq program, the proxy agent pops orders residing on the stack and submits them to the LOB. Before a GP agent executes its program, the proxy agent pushes all outstanding limit orders associated with this agent to the stack. This allows for the GP agent to cancel existing limit orders. *Order* objects on the stack may be inspected. For instance, it is possible for an agent to ascertain the price, size direction (buy/sell), or type (bid/ask) of an object on the *order* stack.

The *market* type provides methods through which LOB state information can be obtained from the proxy trader. For example, the *market* type provides accessor methods to procure the current ask price, bid price, spread, cumulative ask volume, cumulative bid volume, and simulator clock time. Furthermore, state information of the proxy trader is inspectable. The *market* type provides accessors to inspect trader wealth, capital, position (inventory), and utility (as described in section 6.4.1). The actual *market* stack is never utilized. Instructions of the *market* type always push values to stacks of other types. For instance, if the current market price is requested, the value is pushed to the float stack since it is represented by a floating point number. Figure 6.3 illustrates the execution of a short program that utilizes both the *order* and *market* types.

Genetic operators on Staq code

Staq provides genetic operators. Crossover is implemented traditionally: two cut points are chosen and the subtrees are exchanged. Three forms of genetic mutation are employed: cut mutation, point mutation, and point mutation with type matching. In each form of mutation, a cut point is randomly selected. In *cut mutation*, the Staq expression at the cut point is simply deleted. In *point mutation*, the expression at the selected cut point is replaced by a randomly generated expression. For *point mutation with type matching*, the expression at the selected cut point is replaced with an expression of the same type. If the expression at the cut point is a list, a randomly generated list replaces the original expression. However, if the expression at the cut point is an atomic instruction, a new instruction of the same type is randomly selected to replace the original.

6.4 Experimental results

In this section, I report results from two experiments using the GP framework described in section 6.3. Both experiments share the goal of evolving a single trader that achieves consistent profits. The first experiment uses a fitness function based on a known economic utility function. The fitness function of the second experiment incorporates a penalty for the purpose of encouraging inventory control. In both experiments, eight GP runs were conducted. From each set of GP runs, I isolate and describe an individual genetic program that displays behavior representative of the class of agents evolved using the designated fitness function.

Table 6.1 summarizes the general parameters used in both experiments. In Staq, individual stacks and instructions can be disabled. However, for these experiments, all stacks and instructions were enabled. The instruction set contains 247 instructions. A maximum program length of 50 points is enforced. Because loops are possible in Staq, an execution limit is required. The execution limit is set to 100 points. The space of possible source programs is 247^{50}. Because of the enormous search space, a large population of candidate solutions is required. A global population size of 24000 GP agents is employed. The GP agents are distributed across 24 separate islands [62] [63], with each island maintaining a population of 1000 agents. Each island contains its own GP engine and evolves its local population independently. The islands are connected in a ring. At the end of each generation, a group of top agents from each island are chosen to *migrate*, in a clockwise direction, to its nearest neighbor. The migration rate, set to .005, dictates that five agents migrate after each generation. Immigrant agents are automatically promoted to the next generation of their new island. Agents are evolved for 50 generations. During construction of subsequent generations, candidates are selected probabilistically using a round robin tournament of size five.

Table 6.1: General experimental parameters.

parameter	value
Activated stacks (number of instructions)	float (31), int (31), boolean (18), code (47), name (17), order (39), market (20)
Maximum program length	50 points
Execution limit	100 points
Total population	24000
Number of islands	24
Agents per island	1000
Migration rate	.005
Number of generations	50
Round robin size	5
Crossover rate	45%
Mutation rate	45%
Exact reproduction rate	10%
Length of fitness assessment	1000 market timesteps
Noise trader parameters	$\alpha = .071$, $\mu = .150$, $\delta = .0178$ tick size=1.0

6.4.1 Experiment I: Sharpe ratio inspired fitness

The fitness of a trader is related to its ability to accrue profit. However, profit alone does not necessarily define success. Here, I provide motivation for a fitness function based on the economic utility function known as *Sharpe ratio* [50] that integrates the concept of risk aversion.

Profit is integrally linked to *wealth* and many definitions of wealth exist. Three components impact trader wealth: capital, position (inventory), and price. For a given trader i at time t, I denote its capital as c_t^i and its position as x_t^i. Given the midpoint price at time t, denoted as p_t, wealth is defined as $w_t^i = c_t^i + (p_t x_t^i)$. This calculation is approximate since it assumes that a trader's position may be liquidated at the midpoint price, when in fact, a best case scenario for liquidation would take place at either the bid or ask depending on the direction of the position. The term *wealth return*, $r_t^i = w_t^i - w_{t-\upsilon}^i$, refers to the change in wealth over a time interval υ. Wealth returns are used to define the fitness functions to follow. In this context, *time* refers to *simulator* time, as opposed to *event* time.

Fitness function: Sharpe ratio

An economic measure known as the *Sharpe ratio* integrates the concept of profit with that of risk. The measure computes the ratio of average reward to risk. In this context, the average reward is defined as the mean of wealth returns and risk is defined as the standard deviation of wealth returns. Equation 6.1 defines Sharpe ratio, where $\vec{r}$ is the vector of wealth returns for the time period examined, $\langle\rangle$ denotes the mean, and $\sigma()$ represents the standard deviation.

$$S = \frac{\langle \vec{r} \rangle}{\sigma(\vec{r})} \tag{6.1}$$

Used alone, the Sharpe ratio does not provide an adequate fitness function because there is little incentive to trade. For a GP agent to profit, it must first learn to place orders. Not only must it place orders, but it must learn to place orders that yield transactions. Without transactions, a trader cannot profit. To encourage such behavior, GP agents that do not place orders are assigned a minimum fitness value. Agents that place orders but do not achieve transactions are assigned a slightly greater fitness value (but significantly less than the range of typical Sharpe ratio values). By enforcing this multi-plateaued fitness policy, GP agents are first encouraged to place orders. Once this is achieved, the Sharp ratio rewards strategies that yield profitable transactions.

Result: clear & hoist trader

In this section, I describe a particular trader strategy referred to as *clear & hoist* (CH). Within the insulated environment of the simulated market, the strategy successfully optimizes the Sharpe ratio inspired fitness function: within thirty timesteps it achieves infinite wealth. All of the GP experiments using this fitness function discovered some variant of the strategy as the best solution. The clear & hoist trader provides a straightforward example of an evolutionary agent that exploits its environment. The manner in which it achieves success highlights the limitations of the simulated market environment.

As suggested by the name, the CH strategy involves two distinct steps: the *clear* followed by the *hoist*. The strategy preys on the naivete of the noise traders. The detailed mechanics of the strategy are depicted in figure 6.4. The clear order eliminates all existing ask orders on the LOB. The hoist order redefines the best bid price by placing a bid limit order at an inflated price. Following the hoist, noise traders resume trading. As described in section 6.2.2, noise traders place orders relative to the current best bid and best ask prices. Thus, they naively reinforce the hoisted price.

Inspection of the definition of wealth, $w_t^i = c_t^i + (p_t x_t^i)$, reveals that it is not necessary for the trader to sell off its inventory of shares to earn a profit. The inventory contributes to the wealth of the trader. Figure 6.4 depicts a single instantiation of the strategy. However, the process can be repeated *ad infinitum*. With each successful instantiation, the price is ratcheted upwards and the clear & hoist trader secures a profit with minimal risk.

The following Staq program, containing nine points, implements the clear & hoist strategy:

```
(((market_agent_wealth flt_dup) order_constr_lo_bid) (market_volume)
order_constr_mo_bid)
```

The program is a pruned version of that recovered from a GP experiment: auxiliary code that does not directly affect the result was removed. The original program contained forty one points. Figure 6.5 shows the maximum fitness of each generation for the GP experiment that produced this program. The fitness does not increase monotonically. The reason is that each fitness assessment is based on a different random number generator seed. Thus, the fitness results of agents fluctuate. In the sixteenth generation, the fitness approaches infinity. This jump represents the final transition to the clear & hoist strategy.

Execution of the program begins with the `market_agent_wealth` instruction, which pushes the current wealth of the agent to the float stack. The next instruction, `flt_dup` duplicates the value on top of the float stack, thus leaving two copies of the agent wealth value. The `order_constr_lo_bid` instruction pops two float values and constructs a bid limit order. The first popped float value dictates the price and the second determines the size. Next, the program constructs a buy market order. The `market_volume` instruction pushes the current cumulative market volume to the float stack. The following instruction, `order_constr_mo_bid`, constructs a bid market order. The constructor requires a single float value that is used to determine the order size. After executing the program, two newly constructed orders reside atop the order stack. The clear order occupies the top slot; the hoist order rests below. Importantly, the size of the clear order is dictated by the current agent wealth. Agents are provided a starting capital of $10000. With this amount of capital, an agent can create an order for a number of shares sufficient to clear the ask side of the LOB.

This program achieves ultimate success. After buying a significant allotment of shares, the trader inflates the price of the asset. A boost in wealth results. This increase in wealth contributes to even greater profits in subsequent executions of the program. Note that the hoist price is dictated by the wealth of the agent. Since, wealth increases with each instantiation, the magnitude of the hoist increases as well. The positive feedback yields an exponential growth in wealth.

Figure 6.6 presents results from a fitness assessment of this agent. Only twenty timesteps are shown to highlight the dynamics of the exponential boom in both trader wealth and market price. After a saturation period of 200 simulator timesteps, the trader is activated. The top left graph compares the wealth of the CH trader (green) to that of both noise traders. The wealth measurements are relative to the starting wealth of each agent. Note the steep growth at the tail end. Exponential growth quickly follows. Further inspection reveals that the GP agent profits at the expense of the limit order noise trader (purple). The top right graph, shows that the trading volumes of the CH trader and the limit order noise trader are approximately equivalent. The bottom left graph illustrates that the CH trader buys from the limit order noise trader. The large jump at $x = 201$ represents the first *clear* order that buys all ask limit orders that accumulated during the saturation period. The limit order noise trader assumes a negative inventory. The combination of negative inventory and increased price yields diminished wealth. The middle graph illustrates the inventory accumulated by the agents. In the bottom right graph, the logarithm of the bid-ask prices are shown. The prices grow exponentially.

The success of the CH trader is predicated upon two properties of the noise trader population: an incessant willingness to trade and the lack of a perceived value of the traded asset. These conditions do not hold in the real world. Real markets contain traders referred to as *fundamentalists*. These traders associate a perceived value with an asset. Fundamentalists invest in an asset when the market price diverges from their perceived value. As such, their trading causes the market price to revert to the perceived value and would interfere with the CH trader's attempt to hijack the price.

6.4.2 Experiment II: Inventory controlled fitness function

The experiment of section 6.4.1 provides evidence for the efficacy of the GP framework. Next, I conduct a second experiment with the goal of evolving a more realistic trader strategy. First, I opt to update the GP fitness function. Section 6.4.2 introduces a new fitness function, one that incorporates a penalty for poor inventory control. The penalty punishes strategies like clear & hoist, which involve building a large inventory.

Incorporating inventory control

The new fitness function, denoted F', incorporates a penalty for lack of inventory control. The original fitness function, F, provides the foundation for this second generation function. If sufficient inventory control is achieved, $F' = F$. However, if the inventory is not properly managed, a penalty is enforced.

The idea is as follows. If a trader maintains an average inventory less than a given threshold τ, no penalty is incurred. However, if the average position exceeds the threshold, the original fitness F is rescaled. The size of the rescaling is dictated by the magnitude of the infraction. Assume x_t to be the position of a trader at time t. I define $\tilde{x} = \frac{1}{T}\sum_t^T |x_t|$ to be the mean of the absolute values of the positions held by a trader during a specified observation period of T timesteps. If $\tilde{x} > \tau$, a penalty is required.

Equation 6.2 defines the inventory control fitness function, where $sign()$ returns the sign of the supplied variable, and Θ_τ is the unit step function applied at the threshold τ. I refer to the exponential term of equation 6.2, $e^{\Theta_\tau(\tilde{x})\cdot(1-\frac{\tilde{x}}{\tau})\cdot sign(F)}$, as the *scaling term.*

$$F' = Fe^{\Theta_\tau(\tilde{x})\cdot(1-\frac{\tilde{x}}{\tau})\cdot sign(F)} \tag{6.2}$$

The scaling term can be best understood by breaking down the individual components. The $\Theta_\tau(\tilde{x})$ component functions as an activation switch. If $\tilde{x} \leq \tau$, $\Theta_\tau(\tilde{x}) = 0$ and the exponent equals 0. As a result, the scaling term equates to unity and $F' = F$. However, if $\tilde{x} > \tau$, $\Theta_\tau(\tilde{x}) = 1$ and the scaling operation is *activated.* The second term of the exponent, $1 - \frac{\tilde{x}}{\tau}$, dictates the magnitude of scaling. Provided *activation*, it is given that $\tilde{x} > \tau$, and thus $(1 - \frac{\tilde{x}}{\tau}) < 0$. The greater the discrepancy between $\tilde{x}$ and τ, the more negative the term and the greater the penalty incurred. The final term of the exponent, $sign(F)$, dictates the sign of the exponential term and whether the scaling term is greater or less than one: to either shrink positive F values or magnify negative F values. In these experiments, I set $\tilde{x} = 5$ to encourage a small inventory.

Result: market maker

Experiments with F' reveal a consistent solution: *market making.* Traders that implement the strategy are seen in real markets and are called *market makers.* The idea is straightforward: simultaneously buy and sell shares with the qualification that the buy price is lower than the sell price, thus following the proverb "buy low, sell high". Market making takes advantage of impatient traders, such as the noise traders that place market orders.

Figure 6.7 illustrates one version of the market maker strategy evolved by the stack-based GP approach. The green bars represent bid and ask limit orders placed by the market maker. Given that the noise traders place bid and ask market orders in an unbiased manner, there is a roughly equal probability of the bid and ask limit orders of the market maker being cleared through trades. The symmetric placement of limit orders provides the market maker a mechanism for maintaining a balanced inventory.

All of the GP experiments using the F' fitness function discovered strategies resembling market making. Here, I present one of the many variants. The following Staq program is a pruned version of a thirty eight point solution discovered by the GP framework:

```
   (((market_agent_capital) market_price_bid
order_constr_lo_bid) (order_depth ((bool_fromInt)
market_price_ask) flt_dup order_constr_lo_ask_if))
```

The program begins by constructing a bid limit order. The first instruction, `market_agent_capital`, pushes the current capital of the proxy trader to the float stack. The `market_price_bid` instruction pushes the current bid price of the market to the float stack. The following instruction, `order_constr_lo_bid`, creates the bid limit order. Construction requires two float values. The top value on the float stack, representing the current market bid price, dictates the price of the order. The remaining part of the program follows a more circuitous route to create an ask limit order. The `order_depth` instruction evaluates the current depth of the order stack and pushes the result to the integer stack. Due to the presence of the previously constructed bid limit order, a value of 1 is pushed. The `bool_fromInt` argument pops a value from the integer stack, casts it to a boolean value, and pushes the casted value to the boolean stack. Integer values greater than zero are cast to *true*. Thus, following the `bool_fromInt` instruction, the integer stack is empty and a value of *true* resides atop the boolean stack. The `market_price_ask` instruction pushes the current market ask price to the float stack. The `flt_dup` instruction duplicates the market ask price value and pushes it to the float stack. The final instruction, `order_constr_lo_ask_if` creates an ask limit order if and only if a *true* value resides atop the boolean stack. The condition is met and an order is created. Importantly, the price of the ask limit order is set to the current market ask price. The Staq program creates two limit orders: one bid limit order and one ask limit order. The bid limit order is placed at the current bid price. The ask limit order is placed at the current ask price.

Figure 6.8 compares the performance of the market maker GP agent to the two noise traders during a fitness assessment. The GP agent is activated after a saturation period of 200 timesteps. The top left graph illustrates the relative wealth of the traders. The GP trader (green) makes steady profits at the expense of the market order noise trader (orange). In the top right, the volume of trading is compared. The GP agent trades primarily with the market order noise trader and their volumes reflect this interaction. In the bottom left, the inventories of the traders are compared. Note the effect of the inventory control fitness function: the evolved strategy maintains a balanced inventory over the entire period. In the bottom right, the bid-ask prices are shown. The GP market maker strategy stabilizes the bid-ask prices. In contrast to the clear & hoist GP agent, the GP agent maximizes fitness by securing regular, small profits.

By placing orders at the best bid-ask prices, the trader establishes a high probability of having its orders filled. If the orders were placed behind the current best bid-ask prices, the probability of them being filled would be reduced. If the orders were placed in front of the current bid-ask prices, the orders assume a higher probability of being filled; however, the reduced price margin between the orders yields a lower profit for each share to complete a buy-sell cycle. Different market conditions determine the success of the various price placement strategies. An advanced market maker strategy might detect the market conditions and adjust its strategy dynamically. Although different price placement strategies were observed in the GP experiments, none of the evolved strategies dynamically adjusted the price placement based on market conditions. It is possible that with alterations – either by extending the search time, increasing the population size, or exploring the GP parameter space – the evolution of such trading behaviors may be achieved.

In the real world, traders face the additional obstacle of transaction costs. The imposition of such taxes imposes a zone of unprofitability within the existing spread region – thus, further reducing the availability of market making opportunities.

6.5 Summary & conclusions

The research presented in this chapter was motivated by the need to validate the stack-based genetic programming approach for evolving useful trading strategies. Although the clear & hoist (CH) agent of section 6.4.1 does not present a viable strategy for the real world, it does provide evidence of the evolutionary process learning to exploit the simulated market environment. The CH trader exposes a limitation of the simulated market environment: the lack of fundamentalism. Because the trading actions of the noise traders are not influenced by a perceived value, they are easily manipulated by the CH trader. In Chapter 7, I address this shortcoming by introducing two value-based strategies for the continuous double auction (CDA). The second experiment (of section 6.4.2), illustrates how the evolutionary process can be steered by adjusting the fitness function. Neither of the strategies presented in the chapter compare in sophistication to those employed by real traders. However, both outperform the noise traders against which they compete and the solutions do present significant complexity, well beyond simply identifying decision rules for when to buy or sell, as [3]. Economically, the most interesting result pertains to the consistent evolution of market making strategies when using the inventory control fitness function.

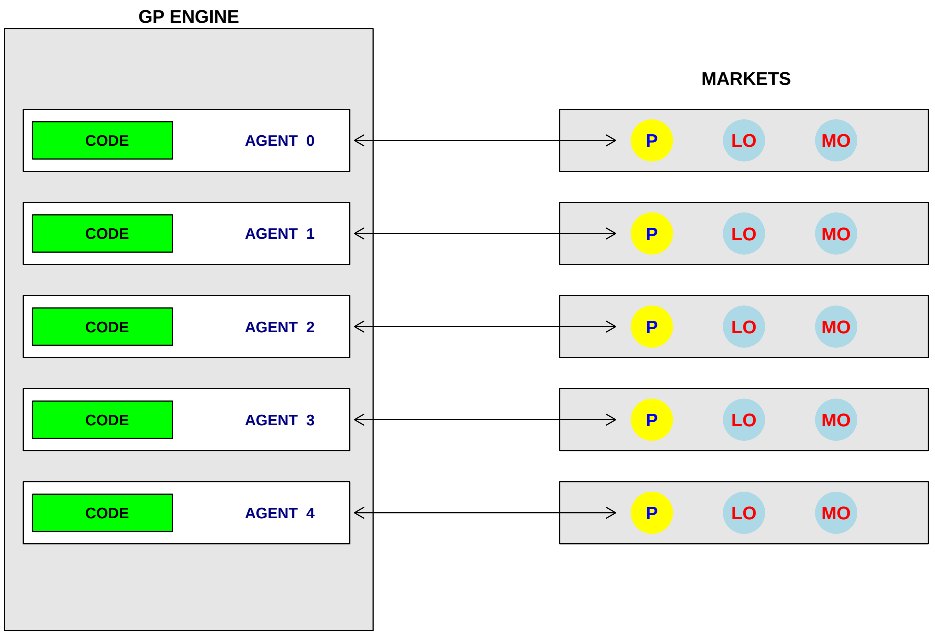


Figure 6.2: Overview of the GP architecture. On the left, the GP engine contains a small population of agents. The fitness of each GP agent is assessed in an independent market (shown to the right). Each market contains three agents: a proxy agent (P), a limit order noise trader (LO), and market order noise trader (MO). The proxy agent provides a gateway for the GP agent to the market.

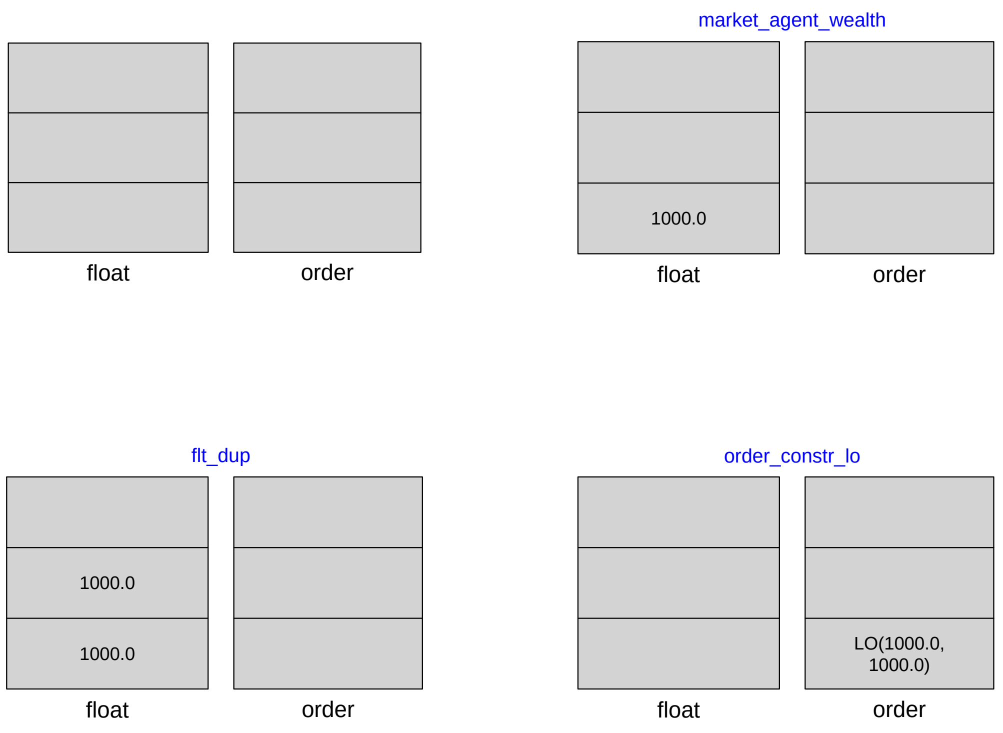


Figure 6.3: Mechanics of a short Staq program that creates a limit order. Two stacks are shown: the *float* stack (left) and the *order* stack (right). The executed program consists of three instructions: `market_agent_wealth flt_dup order_constr_lo`. In the figure, program execution proceeds from left to right, top to bottom. Initially (top left), both stacks are empty. The first instruction, `market_agent_wealth` (top right), pushes a value of 1000.0 to the *float* stack. The second instruction, `flt_dup` (bottom left), duplicates the value residing atop the *float* stack. The final instruction, `order_constr_lo` (bottom right), constructs a limit order. The constructor requires two arguments, both of which are popped from the *float* stack. The first value sets the price of the limit order and the second value dictates the number of shares. After execution of the program, a single limit order resides on the *order* stack.

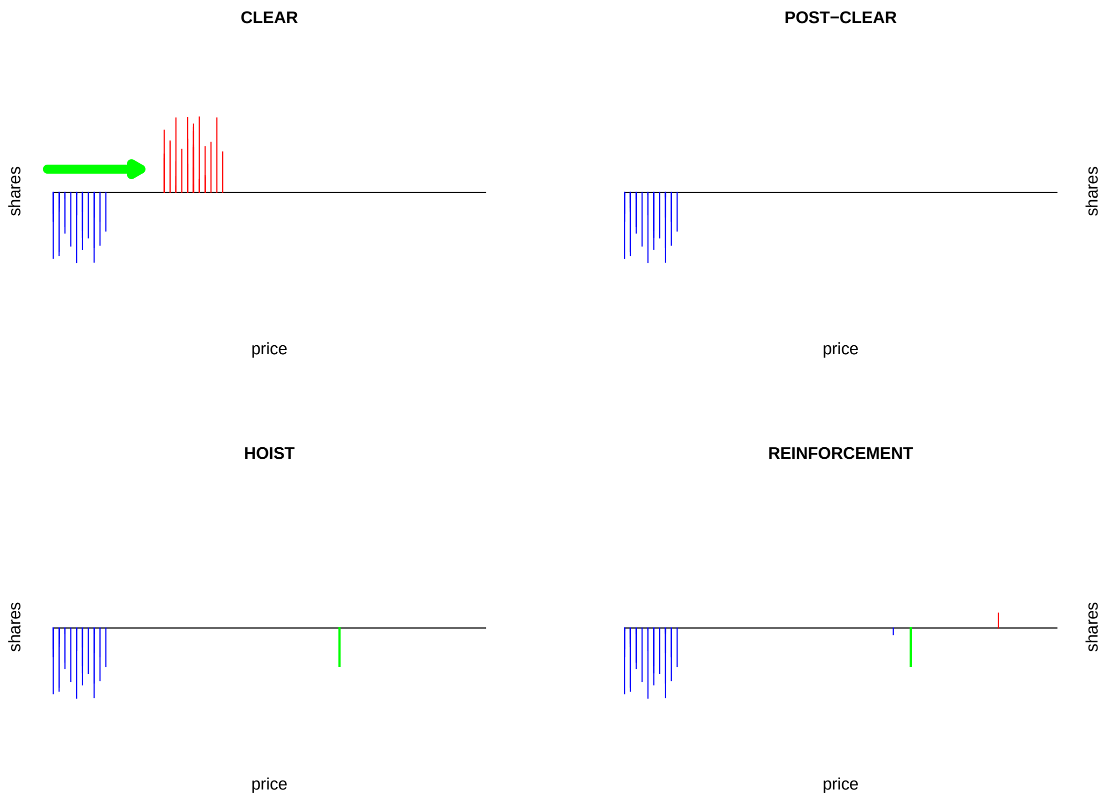


Figure 6.4: Storyboard for the clear & hoist (CH) agent. The snapshots progress in chronological order, from right to left, and top to bottom. The red and blue bars represent limit orders placed by the noise traders. The CH orders are colored green. The agent enters a market saturated with noise trader limit orders. The *clear* involves placing a large buy market order that eliminates all existing ask limit orders (top snapshots). The hoist follows and entails placing a buy limit order at an inflated price (bottom left snapshot). The limit order placing noise trader reinforces the *hoisted* price (bottom right).

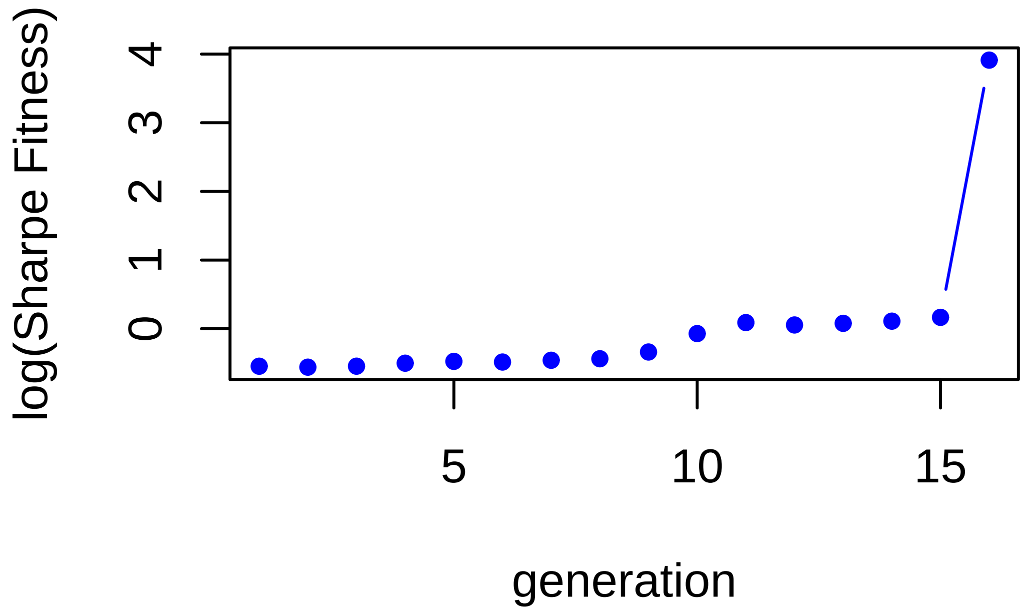


Figure 6.5: GP fitness of the clear & hoist agent. The x-axis represents the GP generation. The y-axis represents the logarithm of the Sharpe ratio fitness. In generation 16, the fitness approaches infinity.

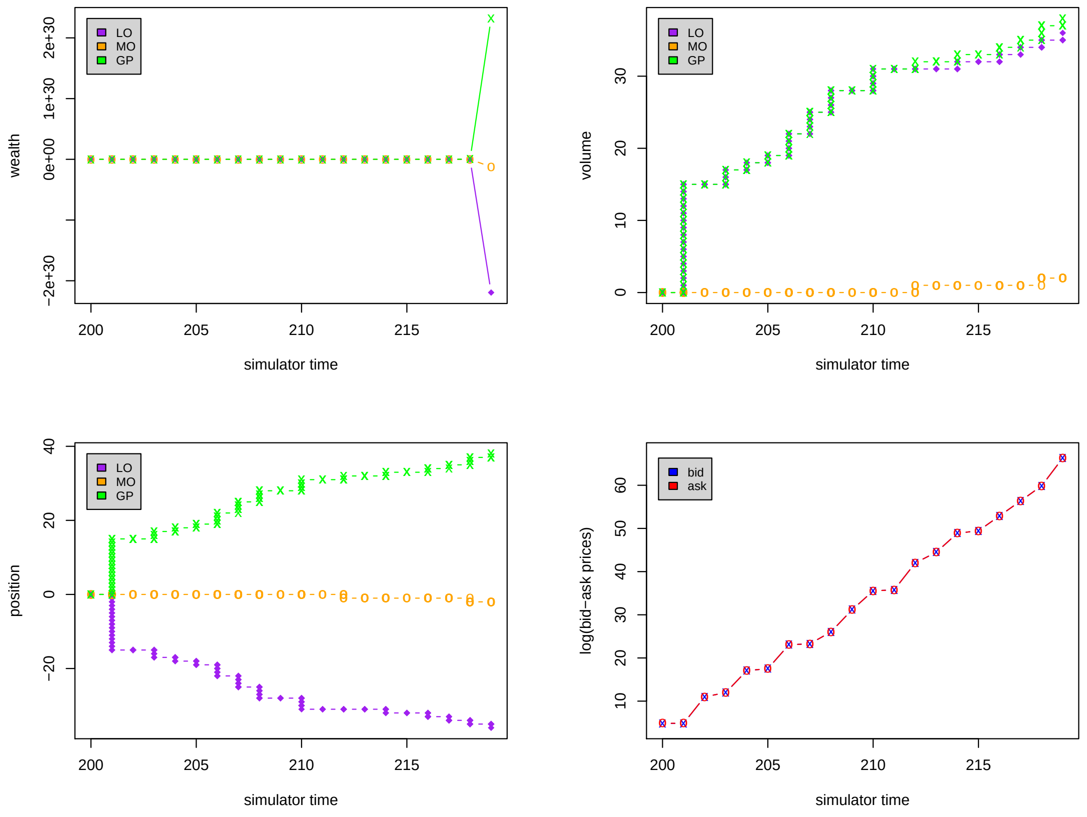


Figure 6.6: Performance of the clear & hoist agent (green). In each graph, simulator time is represented on the x-axis. In the top left, the relative wealth of the GP agent to those of the limit order noise trader (purple) and market order noise trader (orange). The wealth of the GP agent grows exponentially. Volume of trading is compared in the top right. The clean & hoist trader and limit order noise trader trade with one another and share similar rates. In the bottom left, the position of each agent is compared. The CH trader buys from the limit order noise trader. In the bottom right, the log bid-ask prices are plotted. The exponential growth in price inflates the wealth of the GP agent.

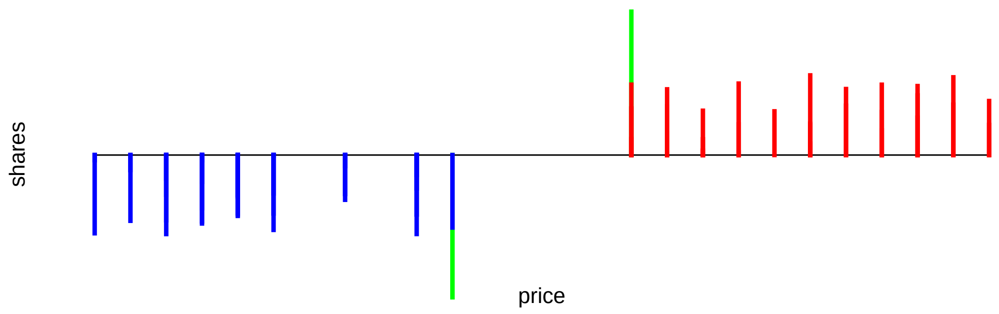


Figure 6.7: Illustration of the market maker strategy. Noise trader limit orders are represented by blue (for buys) and red (for sells) bars. The market maker orders are colored green. In the illustration, the market maker has placed its orders at the best bid-ask prices.

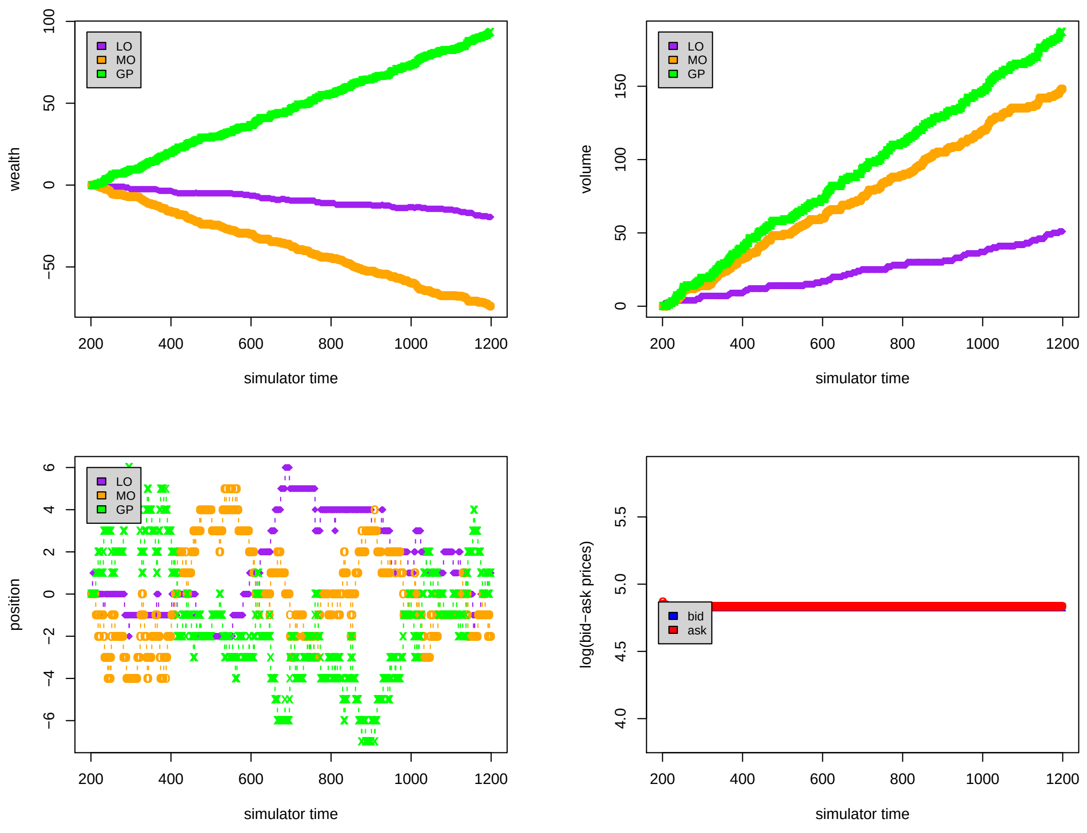


Figure 6.8: Performance of the market maker GP agent (green). The x-axis of each graph represents simulator time. In the top left, relative wealth of the traders is compared. The GP agent makes regular small profits, primarily at the expense of the market order noise trader (orange). In the top right, the volume of trading is compared. The GP agent trades mostly with the market order noise trader. The bottom left compares the relative positions of the traders. Because all of the traders place orders bilaterally in an unbiased manner, they all achieve balanced inventories. The bottom graph shows the bid-ask prices, which are stabilized by the market maker strategy.

Chapter 7

Incorporating fundamentalism

7.1 Motivation

Chapter 6 provided an intermediate step towards coevolving a trader population. There, a single trader that can regularly profit in a market inhabited by noise traders was evolved. Although the clear & hoist (CH) trader of section 6.4.1 represents a dominant strategy in the simulated market environment, it is unrealistic for the real world. In real markets, there exists a subpopulation of traders known as *fundamentalists*, or value traders, that associate a perceived value with the asset being traded. When the market price diverges significantly from its perceived value, a value trader will either buy or sell shares of the asset in anticipation of a price correction. In a market inhabited by noise traders and fundamentalists, it would be impossible for a CH trader to inflate the price of an asset without meeting significant resistance.

This chapter discusses two such value-based mechanisms. The first method involves integrating a fundamental value into the original LONT process. The second method introduces an independent *value trader*. Section 7.2 presents the mechanics of the two value-based methods. Section 7.3 presents experimental results for each of the proposed methods. Section 7.4 compares the merits of each approach and discusses the impact that inclusion of a value-based process has on the GP process.

7.2 Methods

Both value-based traders are based on a dynamic value process, which is introduced in section 7.2.1. In section 7.2.2, I introduce a value-based limit order noise trader process, the VLONT, which resembles the LONT process. In section 7.2.3, an independent value trader (VT) for the continuous double auction (CDA) is described.

7.2.1 Dynamic value process

In real markets, the value of an asset changes over time. As a result, effective value traders continually reassess and update their perceived value of the asset. In a simulated market, the asset is imaginary, and thus has no actual value. I employ a random process to model the dynamic asset value. The value, v, is initially set to the opening price of the market. Thereafter, it is stochastically updated on each timestep with probability π. A random number, $\rho_0 \in [0,1)$, is generated and if $\rho_0 < \pi$, the dynamic value is updated. Given that an update is due, the perceived value is adjusted by flipping an unbiased coin. According to the outcome of the flip, the price is adjusted either one tick upwards or one tick downwards:

$$v_{t+1} = \begin{cases} v_t - \kappa & \text{if } \rho_1 < 0.5 \\ v_t + \kappa & \text{if } \rho_1 \geq 0.5 \end{cases} \tag{7.1}$$

where v_t represents the value at time t, κ denotes the tick size, and $\rho_1 \in [0,1)$ is randomly generated.

In the real world, there exists a diverse collection of value traders, each holding an independent perceived value. In the simulated market environment, I approximate this population with a single value trader that holds a lone perceived value meant to represent the entire subpopulation. Descriptions of the two value-based strategies that make use of this dynamic value process follow.

7.2.2 A value-based limit order noise trader (VLONT) process

In this section, I introduce a value-based limit order noise trader (VLONT) process. In the original limit order noise trader (LONT) model, traders place orders at prices relative to the current bid-ask prices. For bid limit orders, the LONT chooses a limit price from the semi-infinite interval $(-\infty, p_{ask}]$, where p_{ask} represents the current best ask price. Conversely, for ask limit orders, the LONT chooses a limit price from the semi-infinite interval $[p_{bid}, \infty)$, where p_{bid} represents the best bid price. It is because of this choice of boundaries that the CH trader is able to manipulate the LONT. By inflating p_{bid} and p_{ask}, the CH trader is able to increase its own wealth, where wealth is calculated as $w_t = x_t p_t + c_t$ (w_t represents wealth at time t, x_t represents position at time t, p_t represents price at time t, and c_t represents capital at time t). The LONT trader simply accepts the redefined prices and continues to trade.

The VLONT trader functions just as the LONT does, with the exception of the price boundaries employed for the stochastic limit price process. As described in section 7.2.1, the VLONT trader maintains a perceived value. Rather than use the current bid-ask prices as does the LONT, the VLONT uses the perceived value to dictate the price boundaries. Specifically, for bid limit orders, the VLONT chooses a limit price from the semi-infinite interval $(-\infty, v + \omega]$, where v is the perceived value and ω is an offset measured in price ticks. Similarly, for ask limit orders, the VLONT chooses a limit price from the semi-infinite interval $[v - \omega, \infty)$. Figure 7.1 illustrates the difference between how the LONT and VLONT traders choose limit prices.

The VLONT resembles the constrained zero intelligence (ZI-C) trader proposed by Gode and Sunder [20]. Whereas, I approximate the collective behavior of an entire subpopulation of value traders with a single VLONT, Gode and Sunder employ a population of ZI-C traders. Each of these traders maintains a privately held perceived value. The individual ZI-C traders use the perceived value to stochastically generate bid and ask prices – the value dictates either the upper bound for bid prices or the lower bound for ask prices. Unlike the VLONT, the ZI-C do not incorporate a price tick offset ω.

7.2.3 A value trader for the CDA

Here, I describe a value trader (VT) for the CDA. The trader associates a perceived value with the asset being traded. When the market price diverges from this value, the value trader takes action. Otherwise, it remains latent. The rest of this section describes the mechanics of the proposed value trader. Throughout the discussion, we refer to figure 7.2, which illustrates the mechanics of the strategy. Table 7.1 summarizes the value trader parameters discussed.

Role of the bid-ask prices

Many definitions of price exist. As described, the motivation of the value trader is to sell when price exceeds value and to buy when value exceeds price. The best price at which to immediately buy shares is at the ask price, p_a. Thus, when $(p_a < v)$, there exist shares that the value trader considers under-priced and it is motivated

Table 7.1: Value trader parameters

variable	dictates	units	effect
T	activation threshold	price ticks	increased T leads to increased patience
τ	range of shares to buy/sell after activation	price ticks	increased τ leads to decreased order sizes
M	maximum order size	shares	decreased M leads to slower price to value reversion

to buy. Conversely, the best price at which to immediately sell shares is at the bid price, p_b. When $(p_b > v)$, there exists a demand for shares at a price that the VT considers too high and it is motivated to sell. Depending on the situation, I refer to either the bid price or the ask price.

Table 7.2: Thresholds for value trader strategy activation

market	activation	orders placed		bid-ask
shares	trigger	MO	LO	effect
overvalued	$(p_b - v) \geq T$	sell	buy at $\$(v + \tau)$	decreases bid
undervalued	$(v - p_a) \geq T$	buy	sell at $\$(v - \tau)$	increases ask

Trader patience: an activation threshold

An impatient value trader trades whenever price and value diverge. A patient value trader waits for price and value to diverge significantly before taking action. The activation threshold T dictates the patience of the value trader. T is measured in units of price ticks. First, I consider the case of the bid price exceeding value. While $(p_b - v) < T$, the value trader remains inactive. If and when $(p_b - v) \geq T$, the trader sells. The magnitude of the sale is discussed in section 7.2.3. Conversely, in the case of value exceeding the ask price, the value trader remains inactive while $(v - p_a) < T$. However, if the condition $(v - p_a) \geq T$ is met, the trader buys. The larger the value T, the more patient the trader. Table 7.2 summarizes the conditions for activating the value trader. Once either of the thresholds is exceeded, the value trader acts immediately by placing a market order, as opposed to a limit order.

Trader commitment: order size

A second parameter measured in price ticks, τ such that $(\tau < T)$, dictates the quantity of shares to trade after exceeding an activation threshold. The greater the size of the trade, the greater the commitment to the move. When $(p_b - v) \geq T$, the trader places a sell market order equal to the volume of shares residing in the range $R_b = [(v + \tau), p_b]$ (top left of figure 7.2). I refer to this volume as $V(R_b)$. Conversely, when $(v - p_a) \geq T$, the trader places a buy market order equal to the volume of shares, $V(R_a)$ within $R_a = [p_a, (v - \tau)]$ (bottom left of figure 7.2). Generally, τ is chosen such that $(\tau > 0)$. By doing so, the VT only trades shares that it considers over or under valued. The greater the value of τ, the smaller the order size.

Securing a profit

After exceeding an activation threshold and placing a market order, the value trader has established a relative position. To secure a profit, the trader must unwind this position. By choosing $\tau > 0$, the market order placed by the value trader moves the appropriate bid-ask price towards the perceived value but not all the way to it. Based on its perceived value, the VT anticipates that price will continue moving towards v. Motivated by this belief, the value trader places a second order: a limit order in the opposite buy-sell direction of the preceding market order. The size of the limit order matches that of the market order and its price is set to either $v + \tau$ (top right of figure 7.2) or $v - \tau$ (bottom right of figure 7.2) depending on the situation. For each share of the limit order that is traded, a roundtrip profit is secured. The value trader either enacts the proverb "buy low, sell high", or the analogous, "sell high, buy low".

Execution speed

The previous description assumes that the VT can place orders for an unlimited number of shares. Based on this assumption, once activated, the VT can complete its strategy in a single timestep. I introduce a final parameter measured in units of shares, M, to limit the maximum order size for the VT. Recall that the actions of the VT cause price to revert towards the perceived value. The size of the orders placed by the VT effects the speed at which this price reversion occurs. By reducing the maximum order size, the price reversion occurs at a slower rate: the lower the value of M, the longer it takes the VT to complete its strategy. If the volume to be cleared exceeds M, $V(R_a) > M$ or $V(R_b) > M$, the VT is forced to place multiple orders, $N \approx \frac{V}{M}$. For each market order placed, a corresponding limit order of the same size is placed. The VT continues placing orders until all orders within the specified range, either R_b or R_a, are cleared. All new limit orders placed within the specified range prior to completion of the strategy must be cleared as well (these shares can simply be cancelled).

7.3 Results

In this section, I present experimental results of market simulations employing the two value based methods. For all simulations, the noise trader parameters are set to $\alpha = .071$, $\mu = .1495$, $\delta = .0178$, and $\sigma = 30000$ (calibrated to trading of Shell stock on the London Stock Exchange from July 2000 to January 2001). For illustrative purposes, the CH trader employs a linear hoist of 100 price ticks (as opposed to basing the hoist on the current wealth of the trader, which leads to an exponential growth in prices). Section 7.3.1 provides experimental results for the VLONT and section 7.3.2 presents results for the value trader.

7.3.1 Value LONT results

Value vs. price

Figure 7.3 compares the perceived value of the VLONT trader (with $\omega = 3$) to the midpoint price for a simulation containing only the VLONT and MONT. As expected, since the VLONT's choice of limit prices is dictated by the perceived value, the midpoint price and value do not diverge significantly at any given time. The mean distance between midpoint price and perceived value over 50000 simulator timesteps is 1.91 price ticks.

Clear & Hoist Trader vs. VLONT

Figure 7.4 illustrates the performance of the clear & hoist (CH) trader in a market inhabited by the VLONT and MONT. In the top left, the relative wealth of the agents is compared. The wealth of the CH trader (green) oscillates (explained further below). This contrasts the steady growth observed in figure 6.6, in which the limit order noise trader process is based on the LONT rather than the VLONT. The wealth of the VLONT trader also oscillates, but in a negative direction. However, the performance of the VLONT agent is of secondary importance. My chief interest is to ascertain how the VLONT will affect the evolution CH agents. The top right graph compares the volume of trading for the agents. The CH trader trades primarily with the VLONT (purple). The bottom left graph depicts the relative positions of the traders. The CH trader consistently buys from the VLONT trader and builds a positive position. The bid-ask prices are shown in the bottom right graph. The prices depict the struggle between the CH and VLONT traders. The CH trader pushes the prices upwards. The VLONT trader forces the prices downwards – reverting them towards value. The relative wealth of the CH trader (top left), is driven by the price oscillations (bottom right). Because the position (bottom left) of the CH trader grows linearly, the amplitude of the wealth oscillations grow linearly as well. The high variance in the CH trader's wealth reduces its Sharpe ratio (defined as $S = \frac{<\bar{r}>}{sd(\bar{r})}$, where $\bar{r}$ represents an array of wealth returns for the observed period), which is used as the fitness function for the GP experiments. Because of this decrease in fitness, the likelihood of the CH trader being selected for by the GP experiments is reduced.

7.3.2 Value trader results

Value trader vs. noise traders

First, I examine the effect of adding the VT to a market environment containing only the LONT and MONT traders. For the VT, I choose parameters of $\upsilon = 0.1$, $T = 10$, and $\tau = 2$. Unless otherwise stated, $M = \infty$. Figure 7.5 compares the midpoint price (orange) to the perceived value (green) for two different values of M. In both cases, prices track the perceived value; however, for greater M, the mean distance between price and value is reduced.

Next, I examine the effect of adjusting the τ parameter. Two simulations (of approximately $200,000$ market events) containing the VT, LONT, and MONT are conducted with identical parameters except that $\tau = 2$ in one run and $\tau = 8$ in the other. Recall that the increase in τ reduces the size of the limit and market orders placed by the value trader. For $\tau = 8$, the actions of the value trader do not move the price as far away from the activation threshold. As a result, the value trader is activated more often (84% in this case). Another effect of adjusting τ is the mean distance between midpoint price and value. The mean distances are 5.27 and 6.66, respectively, for $\tau = 2$ and $\tau = 8$. In summary, by using a higher τ, the order sizes of the value trader are reduced, the trader is activated more often, and price tracks value with less fidelity. The effect of the T parameter is more straightforward. By reducing the magnitude of T, the activation threshold is lowered and thus price tracks value more closely.

Figure 7.6 provides diagnostics for a simulation inhabited by the value trader (green), the LONT (purple), and the MONT (orange). Both the VT and the LONT profit at the expense of the MONT (top left). Although the value trader does accrue wealth, it does not do so as consistently as the LONT and thus has a lower Sharpe ratio for the observed period. The LONT trades the highest volume; the value trader trades in bursts (top right). These bursts in trading, which synchronize with the most significant changes in price (bottom right) and mark periods of VT activation, increase the VT's wealth return variance and thus decreases its Sharpe ratio. The relative positions of the traders are depicted in the bottom left. The value trader both buys and sells depending on the relation of its perceived value to the bid-ask prices. The bottom right graph illustrates the bid-ask prices: the actions of the VT cause prices to stay within a dynamic range as dictated by both the perceived value v and activation threshold T.

Effect of the value trader on the clear & hoist trader

Figure 7.7 provides diagnostics from a simulation in which the clear & hoist trader (green) is pitted against the value trader (red), LONT (purple), and MONT (orange). In the top left, relative wealth of the traders is compared. The wealth of the CH trader oscillates. These oscillations synchronize with the movement of the bid-ask prices (bottom right). The CH trader steadily builds a positive position (bottom left). Accordingly, when prices swing up, the wealth of the CH agent moves up as well. As the magnitude of the CH's position increases, the magnitude of its wealth returns increases as well. The price oscillations (bottom right) depict the struggle between the CH and VT. Unlike the original scenario in which the VT was not present (see figure 6.6), the CH trader is unable to inflate prices linearly. Rather, the presence of the VT causes prices to revert to the perceived value (downwards).

7.4 Discussion

Without the presence of a value-based trader, the clear & hoist trader accrues unlimited wealth in the simulated market. As shown in section 7.3, both of the value-based processes described in this chapter neutralize the CH trader. The actions of the value-based processes cause prices to revert to value. As a result, the CH trader is unable to inflate prices. Without being able to inflate prices, the CH fails to build wealth consistently.

Both value-based strategies, the VLONT and the VT, successfully cause the midpoint price to track the perceived value. In this paper, the VLONT was used as a replacement for the LONT trader. However, it is possible for both strategies (VLONT and LONT) to coexist in the same environment.

To test the effect that a value-based process has on the GP experiments, I integrated the VT into a market environment inhabited by both the LONT and MONT. To assess the fitness of a candidate GP agent, it is added to the market environment for a prescribed number of timesteps and the Sharpe ratio of the agent is evaluated. Results of the GP experiments reveal that market making strategies are selected by the evolutionary process. Figure 7.8 shows diagnostics from a simulation containing one such GP market maker. According to Sharpe ratio, the GP agent's performance exceeds that of the other traders. The success of the trader is predicated on maintaining a balanced inventory, which reduces susceptibility to large wealth changes driven by price fluctuations, and the attainment of regular roundtrip (buy high, sell low) profits. In turn, the reduced volatility in wealth increases the Sharpe ratio (and, thus, fitness) of the GP market maker. Inclusion of the VT has a major impact on the evolutionary process: the CH trader no longer evolves. Presence of a value-based trader induces large swings in wealth for the CH trader (see figures 7.4 and 7.7). The volatility in wealth reduces the Sharpe ratio (fitness) of the CH trader and thus it is not selected for by the GP process.

Is it possible for a value-based process to evolve on its own? Yes, an agent can simply adhere to a static value that it regards as its perceived value and trade when the bid-ask prices diverge from it. However, it is unlikely that a dynamic perceived value based on a stochastic process would evolve: such a process would add program complexity without lending an apparent fitness advantage given the current simulated market environment. A dividend[1] process is required to increase the likelihood of evolving value-based strategies. With regards to the two value-based strategies discussed, the VLONT strategy is much more likely to evolve than is the VT strategy. First, due to its simplicity, the VLONT strategy can be represented much more compactly given the employed GP instruction set. The possibility of a compact representation increases the probability of its evolution. Second, the VT trader trades in bursts as dictated by the divergence of price and value. The bursty trading yields volatile wealth returns, which in turn reduces the VT's Sharpe ratio. The GP experiments employ a Sharpe ratio fitness function. As a result of its bursty trading, the fitness of the VT strategy is inherently challenged and selection of the strategy is thus unlikely.

With regards to coevolving an entire trader population, the role of a value based

[1]A payment made to shareholders.

process raises interesting questions. By including the VT amongst the base organisms (MONT and LONT), the evolution of strategies like the CH will be inhibited. However, one can investigate whether the coevolutionary process uncovers a value-based process on its own. In the coevolutionary scheme, it is no longer necessary for an agent to dominate others to survive. Rather, if the agent could maintain small profit margins, it could endure. Thus, there exists the possibility of coevolving a value based strategy. This presents an interesting hypothesis regarding the coevolutionary process: do value-based processes coevolve intrinsically?

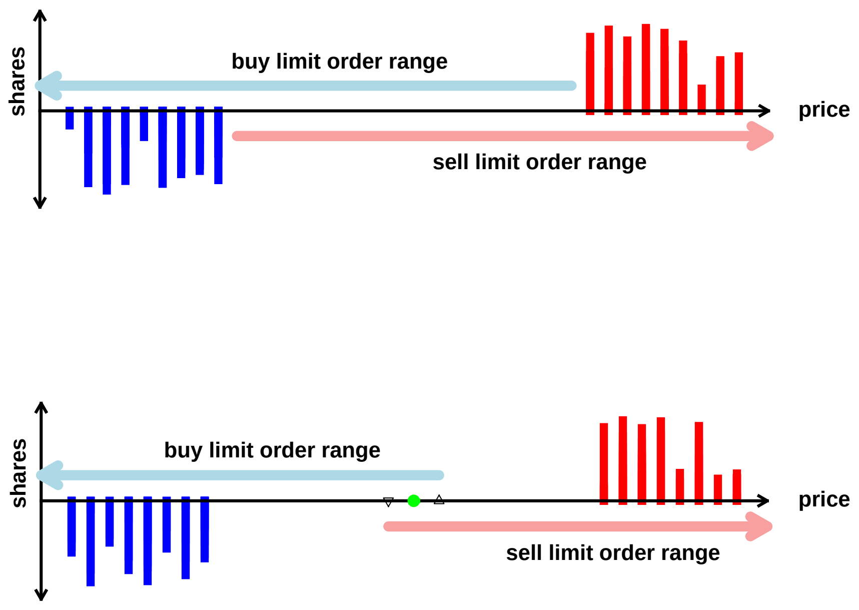


Figure 7.1: Comparison of how the LONT and VLONT traders choose limit prices. Both models draw limit prices from semi-infinite intervals. However, the boundaries of these intervals differ. The top illustration depicts the LONT model, which uses the current bid-ask prices as the boundaries. The bottom illustration depicts the VLONT model. The perceived value of the VLONT trader is marked with a green dot. The VLONT trader applies an offset to the perceived value to dictate the boundaries (marked with black triangles) of the semi-infinite interval from which limit prices are chosen.

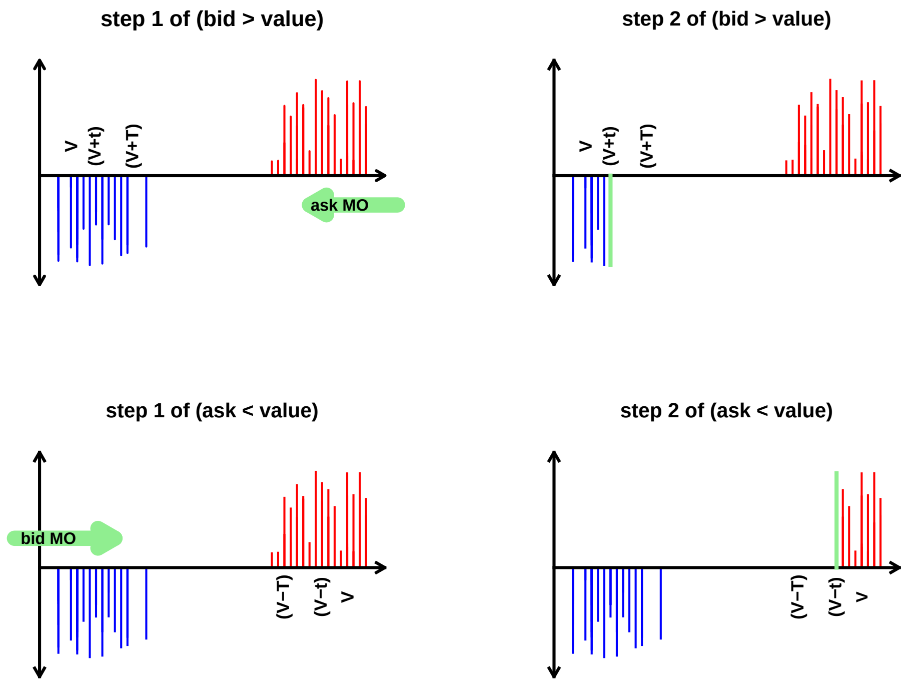


Figure 7.2: Mechanics of the value trader (VT) strategy. The top two illustrations refer to the situation where price exceeds value, $(p_b > v)$. Initially, the VT places a sell market order equal to $V(R_b)$, the volume of shares within the range $R_b = [(v + \tau), p_b]$ (top left). It then places a buy limit order of the same size at a price of $v + \tau$ (top right). The bottom two illustrations depict the actions of the VT for the situation where value exceeds price, $(p_a < v)$. First, the VT places a buy market order for a volume $V(R_a)$, where $R_a = [p_a, (v - \tau)]$ (bottom left). Subsequently, the VT places an ask limit order of the same size at a price of $(v - \tau)$ (bottom right).

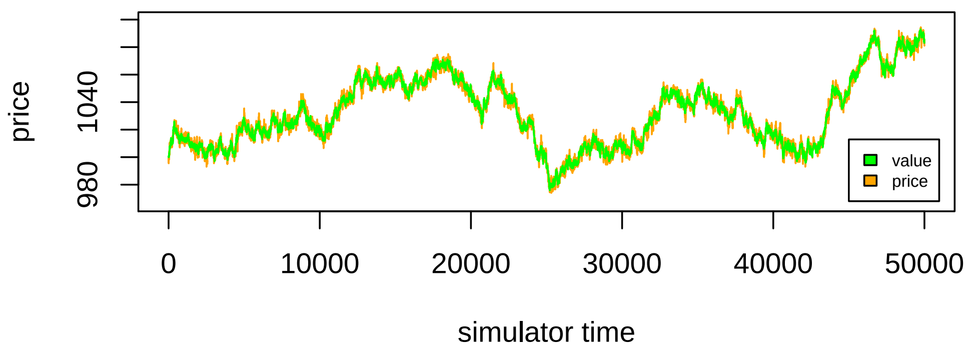


Figure 7.3: Value vs. price for the VLONT trader (with $\omega = 3$). Midpoint price (orange) tracks the value (green) of the VLONT trader. The mean distance between midpoint price and value is 1.91 price ticks.

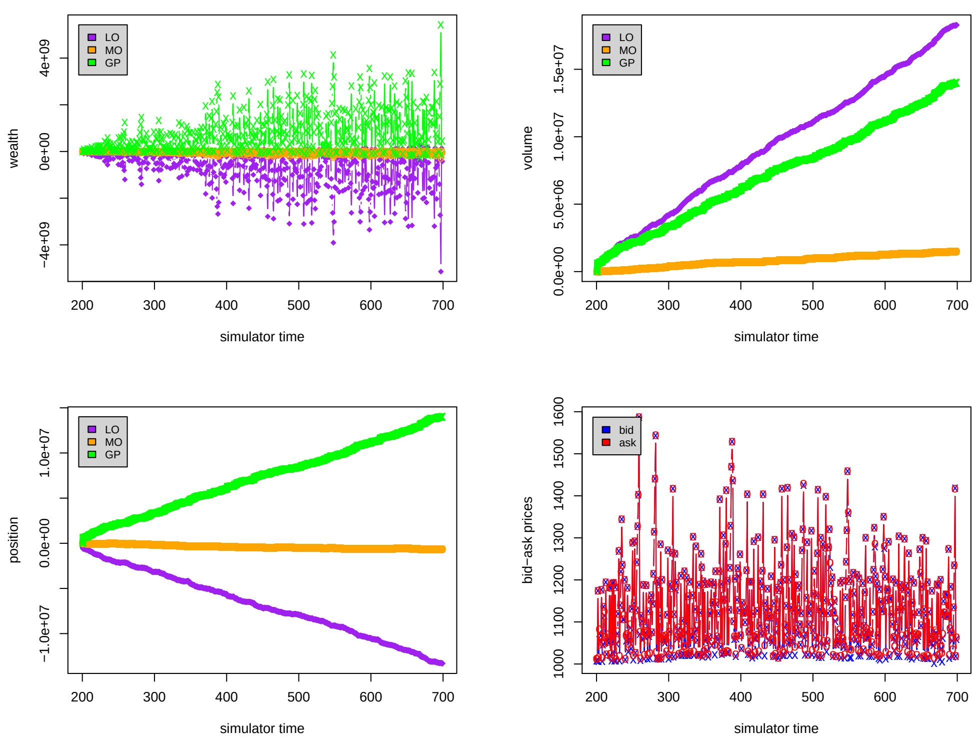
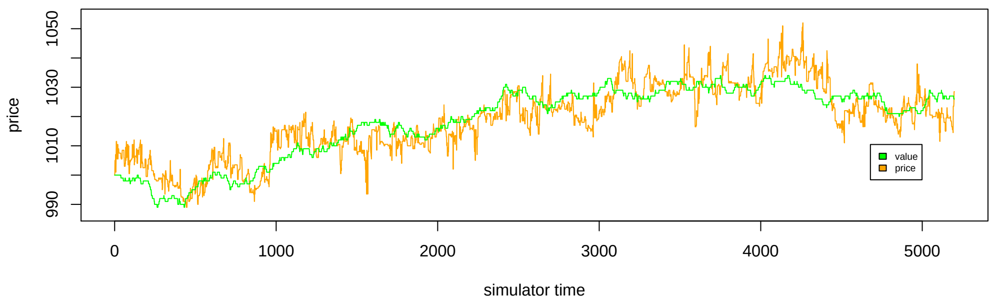


Figure 7.4: Clear & hoist (green) trader in a market inhabited by the VLONT (purple), and MONT (orange). In each graph, the x-axis represents simulator time. The graphs compare relative wealth (top left), volume of trading (top right), position (bottom left), and bid-ask prices.



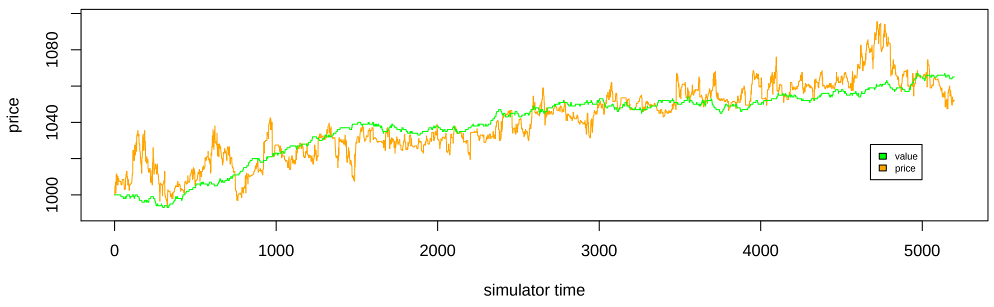


Figure 7.5: Comparison of midpoint price (orange) to perceived value (green) for the value trader (VT) with $\tau = 2$ and two different values of M. In the top graph, the maximum order size is unlimited, $M = \infty$, and the mean distance between price and value is 5.09 price ticks. In the bottom graph, the maximum order size is limited to $M = 6000$, which is equal to 20% of the static order size employed by the noise traders. The mean distance between price and value grows to 7.76 price ticks. In both cases, price tracks value. For larger M, the tracking is of greater fidelity.

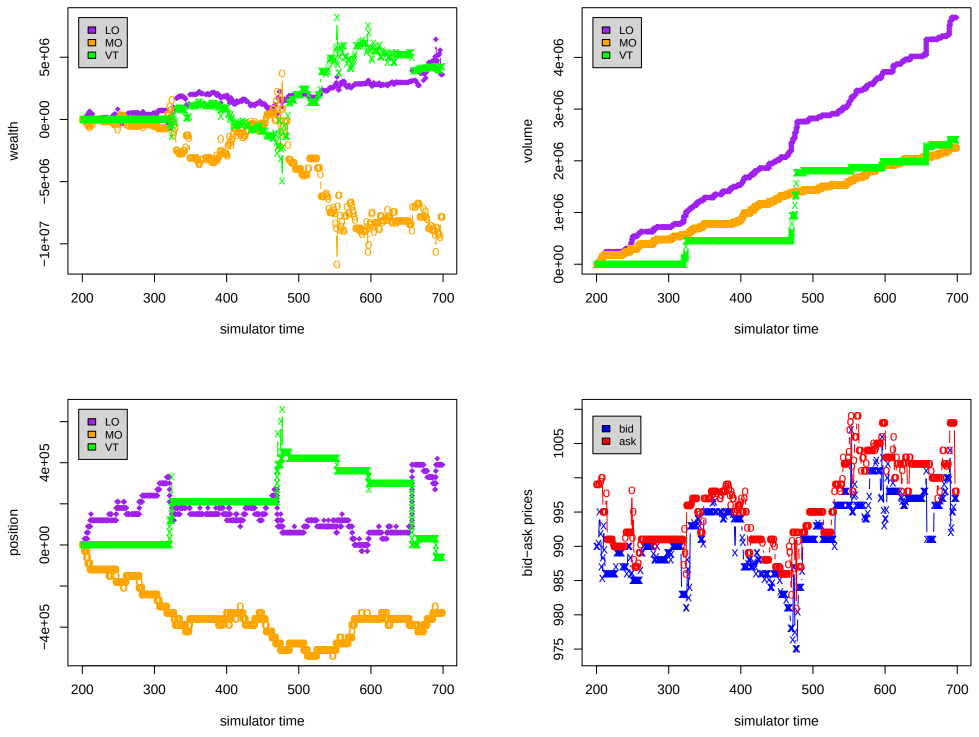


Figure 7.6: Performance of the value trader (green), with $\tau = 2$, in a market inhabited by the LONT (purple) and MONT (orange). In all graphs, the x-axis represents simulator time. Relative wealth is compared in the top left; volume in the top right; position in the bottom left; and, the bid-ask prices in the bottom right. Based on Sharpe ratio, the LONT ($S = 0.0317$) outperforms the VT ($S = 0.0213$).

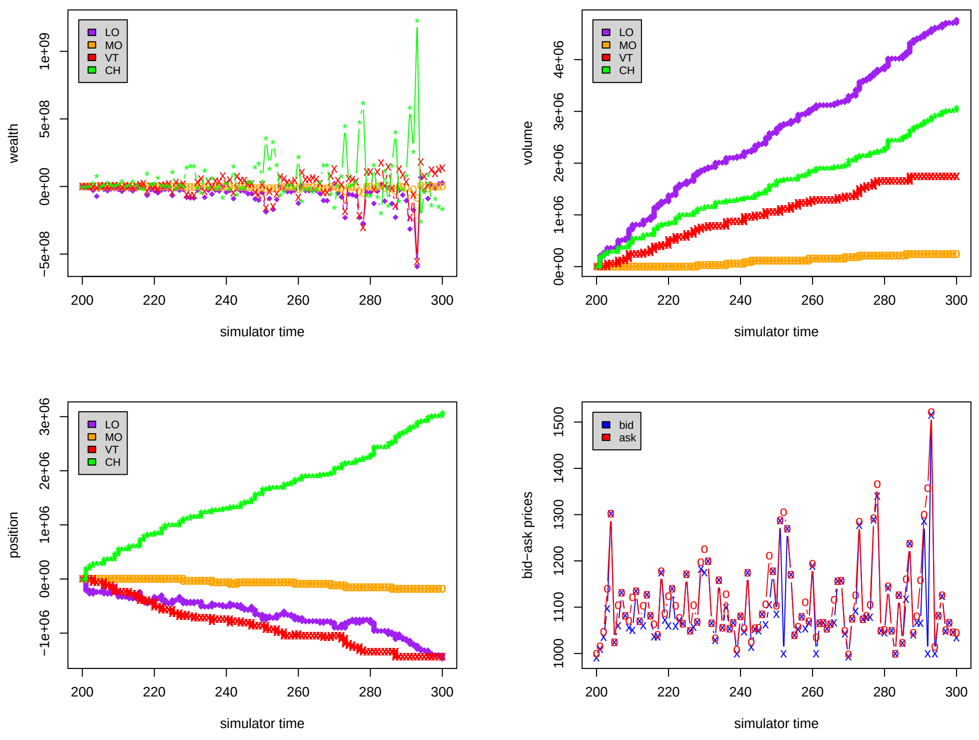


Figure 7.7: Clear & hoist (green) performance in a market environment inhabited by the VT (red), LONT (purple), and MONT (orange). In all graphs, the x-axis represents simulator time. Relative wealth is compared in the top left; volume in the top right; position in the bottom left; and, the bid-ask prices in the bottom right.

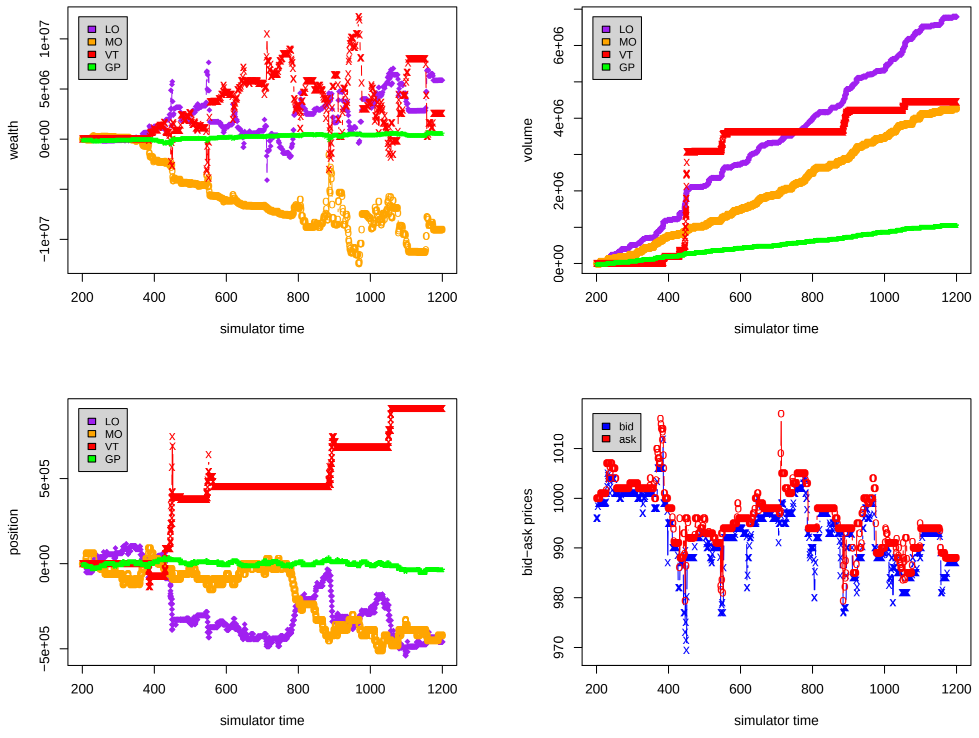


Figure 7.8: Performance of a GP evolved market maker (green) in an environment inhabited by the LONT (purple), MONT (orange), and VT (red). In each graph, the x-axis denotes simulator time. In the top left, the wealth of the agents is compared. The GP agent makes small, regular profits. In the top right, trading volumes are compared. The GP agent trades a small volume in comparison to the other traders. The VT trades in bursts. In the bottom left, the positions of the agents are compared. The GP market maker maintains a relatively balanced inventory and thus is not prone to large changes in wealth driven by price fluctuations. By attaining regular, small roundtrip profits and balancing its inventory, the GP market maker achieves the highest Sharpe ratio.

Part III

Recovery of trophic species in financial market data using community detection

%

Chapter 8

Recovery of trophic species from financial market data

In this chapter, I use the tools developed in part I with the objective of detecting trophic species in financial market data. Ecologically, *trophic species* are functional groups containing organisms that both consume and are consumed by identical species within a food web. Trophic species differ from taxonomic species, which are grouped according to morphology. For the purpose of this work, I consider a trophic species within a financial market to be a group of traders that exhibit temporally correlated functional behavior. For instance, traders that tend to buy and/or sell at similar times meet this criterion. This definition differs slightly from the ecological definition, which ignores temporal associations and focuses on the flow of energy between species.

This chapter progresses as follows. In section 8.1, the methods used for detecting trophic species in financial market data are introduced. In addition, I discuss a methodology for assessing the significance of the detected structure. In section 8.2, I examine the efficacy of the detection methods on two types of data. First, I demonstrate that the methods successfully detect different trader types in data generated by the simulator of Chapter 6. Here, no GP traders are used – all of the trader types are hard-coded. Second, I examine a high liquidity stock (Shell) of the London Stock Exchange (LSE), detecting significant structure in several month-long segments of the real-world data. However, significant structure is not detected in all of the months. Section 8.3 provides a discussion.

8.1 Methods

8.1.1 Temporal detection of trophic species

My method for the detection of temporal-based trophic species involves the following steps:

1. Splice market data into time intervals.
2. Define indicators to be examined.
3. Calculate *trader similarity* between each pair of traders.
4. Create a complex network based on the trader similarities.
5. Apply the deterministic division (DD) community detection algorithm (of section 3.2.1) to the complex network.
6. Test the significance of the detected communities.

Below, I describe each step of the process in greater detail. I address step 6 in section 8.1.2.

Splice market data into time intervals

I splice the provided financial market data into time intervals. Each interval contains a specified number of market events, consisting of one of the following: order placement (market or limit), the deletion or expiration of a limit order, or a transaction (trade) between two parties. Two parameters exist for time intervals: First, the total number of events, denoted E, and the length of each interval window, denoted W. Holding E steady, a decrease in W yields a greater quantity of intervals. Holding W steady, an increase in E yields a greater number of intervals. In section 8.2, I vary these parameter settings and analyze the resulting effect.

Defining indicators

To assess the functional similarity of traders, we must identify behaviors to examine. I concentrate solely on two trader actions: buying and selling. I refer to each of these behavioral descriptors as *indicators*. Mathematically, an individual indicator n for trader i is written as $I_i^{n,t} \in \{0,1\}$, where t indicates the time interval observed. If a given trader i exhibits the specified behavior n during a particular time interval t, $I_i^{n,t} = 1$. Otherwise, $I_i^{n,t} = 0$. This is implemented by creating a matrix for each indicator. Each column of the matrix represents an independent time interval. Each row represents an independent trader. Thus, given m traders and n time intervals, the matrix will have the dimensions m by n.

Although I focus here solely on buy/sell actions, a multitude of indicators are possible. For example, we could increase the specificity of the buy/sell indicators and examine separately the placement of market orders and limit orders – thus, creating four independent buy/sell indicators. Or, we could examine the temporal patterns of order deletions. Note that the use of independent buy and sell indicators differs from [31] [65], where a single indicator is employed to indicate the change in inventory (Zovko uses the sign of the inventory change rather than the actual difference).

Trader similarity

Having defined the indicators, we next consider how to measure the functional similarity between traders, defining *trader similarity*, denoted S_{ij} for traders i and j. The measure is built upon a normalized variant of *mutual information*, which measures the mutual dependence of two random variables. Mutual information is defined as:

$$M(X,Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right) \tag{8.1}$$

.

Here, assuming boolean indicators, $p(x) = p(I_i^n) = \frac{1}{T} \sum_t I_i^{n,t}$, where T represents the number of time intervals. Likewise, $p(y) = p(I_j^n) = \frac{1}{T} \sum_t I_j^{n,t}$ and $p(x,y) = p(I_i^n, I_j^n) = \frac{1}{T} \sum_t \left[(I_i^{n,t} == 1) \& (I_j^{n,t} == 1) \right]$. If the variables X and X are independent, the mutual information is zero. Unlike the Pearson correlation coefficient, which measures only linear correlation, mutual information captures both linear and nonlinear dependencies between variables.

Symmetric uncertainty [64] provides a normalized measure of mutual information and is defined as:

$$U(X,Y) = \frac{2 \cdot M(X,Y)}{H(X) + H(Y)} \tag{8.2}$$

where $H(X)$ represents the marginal entropy of the random variable X. I define trader similarity between traders i and j as the mean symmetric uncertainty over the set of indicators:

$$S_{ij} = \frac{1}{|\{I\}|} \sum_{I^n \in \{I\}} U(I_i^n, I_j^n) \tag{8.3}$$

.

Trader similarity provides a generalized measure of functional likeness. Additional indicators could easily be incorporated by appending to the set $\{I\}$. Again, the inclusion of multiple indicators differentiates my approach from both [31] [65]. The higher the value of S_{ij}, the greater the functional similarity between traders i and j. Trader pairs exhibiting independent indicator patterns yield a trader similarity of $S_{ij} = 0$. Because mutual information measures the dependence of two variables, and not their correlation, it is possible that trader strategies whose indicators are polar opposites will be grouped together. However, identifying such differences is trivial. A new trophic species should be created to represent each independent strategy type.

Creation of complex network from trader similarities

From the trader similarity matrix, I create a complex network. Each trader is represented by a node, and the weights of edges between nodes indicate the trader similarity between the nodes. The trader similarity network is fully connected since it is possible to compute similarity between any two traders.

Detection of trophic species

To detect temporal-based trophic species, the DD community detection algorithm (presented in section 3.2.1), using the original modularity measure, is applied to the complex network. Because of the resolution limit explained in Chapter 4, it is necessary to apply the DD algorithm recursively to each detected community until the sub-communities are deemed indivisible. Before doing so, the sub-network associated with the community is extracted from the original network. As an alternative, to overcome the resolution limit inherent to modularity, it is possible to optimize divisionality using the same algorithm.

8.1.2 Testing the significance of the detected trophic structure

The algorithmic process of 8.1.1 proposes a set of trophic species. In this section, I propose methods for both validating the detection process and testing the statistical significance of the detected trophic species.

Communal overlap in the presence of known structure

Using simulated data, we may possess knowledge of the trophic species *a priori*. In this case, we can validate the trophic detection algorithm by comparing the known trophic species to the detected species using the measure of communal overlap presented in section 5.2.3. The higher the communal overlap value, the higher the detection fidelity. Of course, without knowledge of the true underlying trophic species, it is not possible to measure communal overlap and validation is impossible. Thus, we require a different method to test the significance of the detected trophic species when using real-world financial market data.

Test of statistical significance using surrogate data

I propose the following method, using surrogate data and a rank-based test [57], for evaluating the significance of the detected trophic species when the true communities are not known.

1. Create original complex network (denoted N_o), apply the community detection algorithm, and measure the divisionality of the detected communities (denoted D_o).
2. Establish an α significance level such that $R = \frac{1}{\alpha} - 1$ surrogate networks are required.
3. Create surrogate network N_r by stochastically switching edge weights of the original network N_o. Γe switches are performed, where $\Gamma = 100$ and $e = m(m-1)/2$ represents the number of edges in the network.
4. Apply the community detection algorithm to the surrogate network N_r.
5. Measure the divisionality, denoted D_r, of the detected communities.
6. Increment r.
7. If $r < R$, go to step 3. Otherwise, continue.
8. Rank the set of divisionality values consisting of D_o and D_r for all $r \in R$.
9. Test hypothesis: is the divisionality value associated with the original network, D_o, the largest?

The objective is to determine whether the community structure represented by N_o is significantly different than that observed in networks with randomly switched trader similarities. If so, I assert that the temporal trophic structure observed in the original data is significant. I adopt divisionality, demonstrated in Chapter 4 to provide a fine-granularity measure of community structure, as an observable measure. By randomly switching the edge weights of N_o to create surrogate networks, the edge weights are redistributed, yet their actual quantities are maintained and the cumulative edge weight of the network is preserved. As a result of this preservation, I consider the relative comparison of divisionality values from the different networks to be permissible[1]. By ranking the set of divisionality values and asking whether D_o is the largest, the objective is achieved. I assess the null hypothesis stating: the community structure in N_o is not significantly different than that observed in networks with randomly switched edge weights. If $D_o > D_r$ for all $r \in R$, the null hypothesis is rejected.

The α value determines the level of significance for our test. Note that if $\alpha = .05$, as is the case in the experiments of section 8.2.2, the required number of surrogate data sets is $R = 19$. If the original data set is not significantly different than the surrogates, there is an α probability that it will randomly have the highest divisionality value. In this case, the null hypothesis will incorrectly be rejected.

8.2 Results

In this section, I present the results of applying the trophic detection algorithm to both simulated and real-world financial market data.

[1]In general, comparing divisionality values computed on networks with different edge weights is not advised. The same holds for modularity. However, a meaningful observable is required for the rank-based test. Comparing a divisionality value against those of networks with switched edge weights, though liberal, provides a gross means of comparison, at the very least. Recall from section 4.2.1, that the local D_i measures compute the ratio of observed edge weight encapsulated by a community to the expectation of this value. In a network with randomized edges, it is expected that $D_i \approx 1$. Thus, if the global divisionality value of the original network does not significantly differ from those of the surrogates, I deem the community structure in the original network insignificant.

8.2.1 Trophic detection on simulated data

Description of data

Using the simulator introduced in Chapter 6, I generate artificial financial market data. The goal of this experiment is to generate data with known trophic species, run the trophic detection algorithm on the generated data, and compare the detected trophic species to the known structure. Towards this goal, I introduce several different hard-coded trader strategies to the simulator. Each trader added to the simulation is given a unique integer identifier (ID). The identifiers are sequentially assigned to traders sharing the same strategy. This ID scheme is helpful for visualization purposes in section 8.2.1.

Three types of *noise traders* are included: the MONT (IDs $3-5$), the LONT (IDs $6-8$), and a combination noise trader (IDs $0-2$) that places both market and limit orders. Three independent traders of each type are included. Their cumulative order flows are calibrated to approximate those observed for the Shell market of the LSE during the month of July 2000 ($\alpha = .071$, $\mu = .1282$, $\delta = .00838$, $\sigma = 33279$, and a price horizon of 28.9).

In addition, I add a *trend trader* type that bases its actions on the last χ price movements. Price movements are represented by the set $P = \{DOWN, SAME, UP\}$. Traders actions are represented by the set $A = \{BUY, HOLD, SELL\}$. The trend trader exclusively places market orders with a size of one share. The size of the strategy space for a trend trader is $|A|^{(|P|^{\chi})} = 3^{(3^{\chi})}$. Table 8.1 provides an example strategy matrix for a trend trader with $\chi = 2$. For the simulated run, I set $\chi = 2$, which yields a strategy space of $3^{(3^2)} = 19683$ possibilities. I incorporate 27 randomly created trend trader strategies into the simulated market. For each of these selected strategies, three independent trend traders are added – yielding a total of 81 trend traders. The trend traders comprise IDs $9-89$.

Each sequential set of three IDs represents a different trend trader strategy.

In Chapter 6, we saw that the GP agents commonly evolved *market maker* strategies. To reiterate, a market maker is a trader that simultaneously places buy and sell limit orders with the intent of making a roundtrip profit. I include hard-coded market makers in this simulation. The only parameter for these traders is the size of the spread required to activate the strategy. Three different spread activator thresholds are used (2, 10, and 20 price ticks). When the given threshold is reached, the trader begins to play both sides of the market. It continues to do so until the spread is reduced below the given threshold. While activated, if either of its orders (buy or sell) is transacted upon, the trader immediately replaces it with a limit order at the current bid or ask. For each selected threshold value, four independent market makers are added to the simulation (IDs $90 - 101$).

The final trader type is the *value trader* presented in section 7.2.3. Three value traders are added, each with parameter settings of $T = 10$ and $\tau = 2$ (IDs $102 - 104$). In total, as indicated by the ID numbers, 105 traders are added to the simulation.

Table 8.1: An example strategy matrix for a trend trader based on the last two price movements.

Price movement at time		
t-1	t-2	Action
DOWN	DOWN	BUY
DOWN	SAME	BUY
DOWN	UP	HOLD
SAME	DOWN	SELL
SAME	SAME	HOLD
SAME	UP	HOLD
UP	DOWN	SELL
UP	SAME	HOLD
UP	UP	BUY

Results for simulated data

Figure 8.1 provides two-dimensional community plots comparing the detected trophic species to the known trophic species, as determined by underlying trader types. For this analysis, $E = 20000$ and $W = 100$. The top plot relates to the original data set; the bottom plot represents a surrogate data set. Points on the x-axis of each plot represent the *known* trophic neighbors (by ID) for each trader (including itself). Points on the y-axis represent the IDs of the *detected* trophic neighbors for each trader. Recall that traders with the same strategy are given sequential ID numbers. Each trader strategy type is colored differently. However, traders of the same type but different parameter settings (or strategy matrices in the case of trend traders) are represented with the identical color. Because of the sequential ordering of IDs, if trophic species are detected with high fidelity, the majority of points should be located on the diagonal.

The top plot exhibits this diagonal pattern. The only exception occurs in the ID range of $0 - 8$. The noise traders have been split into three communities; however, these partitions do not match the known noise trader types. This result is not entirely surprising since limit orders and market orders are not differentiated, and these traders place orders according to a stochastic process. Despite these factors, all noise traders have been grouped with other noise traders. The cyan points represent the trend traders. Notice that for each possible trend trader strategy, the traders have been properly grouped into a trio. Furthermore, the differently parameterized market makers (royal blue) are correctly grouped into separate groups. The fundamentalists (green) are properly detected as well. The communal overlap value for the original data is $\Omega_o = 0.943$.

The bottom plot, that relating to a surrogate data set, differs greatly. The thin line of points on the diagonal exists because each trader is included in both its known and detected trophic communities. However, notice the scattering of points away from the diagonal. In the surrogate case, traders are not grouped with like strategies. The communal overlap value for the surrogate data is $\Omega_s = 0.20$. Figure 8.1 provides visual evidence that the trophic detection process provides high fidelity recovery on the original simulated data. Further, the community structure present in the original data is clearly disrupted by the switching process used to create the surrogate data sets.

For the sake of comparison, I examine the same data set with the same E and W parameter settings, but with a new set of indicators. For this study, I employ a single indicator representing the *net trading* of a given trader (as done by Zovko and Farmer [65]). This indicator can assume one of three values, $I_i^{net} \in \{-1, 0, 1\}$. The $I_i^{net} = 0$ value indicates that the given trader either bought as much as it sold or it was inactive during the period. Otherwise, the sign of the indicator reflects either net buying or net selling. The communal overlap value drops to $\Omega_o = 0.849$ using the net trading indicator (compared to $\Omega_o = 0.943$ for the buy/sell indicators). Comparison of the top plot in figure 8.2 to that of 8.1 illustrates the reason for the decrease. Classification of the market maker strategies (the blue points in the ID range of 90-101) has deteriorated. Recall that three different groups of market makers exist, each with a different spread activator threshold. For each threshold, four independent market makers are added to the simulation. When using the buy/sell indicators, all three groups of market makers are correctly identified as separate, independent entities. Using the net trading indicator, the picture is more muddled. Here, market makers with different activation thresholds are grouped together (demonstrated by the broad band of blue points). Further, two market makers are separately combined with different groups of trend traders (cyan points). More innocuously, a MONT (red points) is grouped with a set of market makers sharing different spread activator thresholds.

By independently assessing buying and selling, the detection of *market making* strategies is improved. Simply tracking the change in inventory does not relay as much information. If there is an imbalance in trading during an interval, the sign will indicate either *buying* or *selling*. Further, if a trader buys and sells an equivalent amount during a given interval, there is no change in inventory. This situation is impossible to differentiate from one in which the trader was inactive.

Next, I examine the effect of varying both E, the number of events analyzed, and W, the length of the time interval window, on the trophic detection process. Figure 8.3 provides three plots with different E values. The top plots relates to $E = 10000$, the middle plot to $E = 20000$, and the bottom plot to $E = 40000$. For each plot, the x-axis represents W ranging from $W = 25$ to $W = 500$ with increments of 25 events. The left y-axis, associated with the triangular points, represents communal overlap. The right y-axis, associated with the circular points, represents the Z-score of D_o (computed over the set of values including D_o and D_r for all r). Each point represents the mean value for 10 separate analyses. For each of analysis, the time intervals are shifted by a random offset with a maximum offset of 10000 events. The offset changes the boundaries for all time intervals in given data set. By conducting multiple analyses, results are generalized and not dominated by the peculiarities of any particular data sequence.

Examination of the top plot in figure 8.3, associated with $E = 10000$, demonstrates that high fidelity detection is possible with a wide range of W values. For $W \in [25, 375]$, communal overlap meets or exceeds $\Omega = 0.8$ and the mean Z-score of D_o meets or exceeds 4.0. Beyond this point, the values tail off, yet $Z(D_o) > 1.0$. Comparison of the top plot to the middle and bottom plots reveals that an increase in E yields more consistent values. This result is not surprising. The functional behavior of traders do not change over time in the simulated environment. Thus, by increasing the length of the data set and thus increasing the number of intervals to analyze, higher fidelity detection is possible. In a real-world market, this relationship may not hold because the functional behavior of traders is expected to be less consistent. As a result, an increase in E may not lead to an increase in detection fidelity. The middle and bottom plots support the claim that high fidelity detection can be achieved over a wide range of W lengths and demonstrate the robustness of the temporal trophic detection algorithm when the functional behavior of traders is consistent.

8.2.2 Trophic detection on LSE Shell data

Description of data

The real-world data that I analyze is extracted from the high-liquidity Shell market of the London Stock Exchange (LSE)[2]. In general, each trader (firm) represented in this data acts on the behalf of many individuals and institutions having different strategies. As such, the definition of “trader” in this data is rather crude. Each trader is associated with a unique identifier (ID). However, to obfuscate trading patterns, the LSE assigns new identifiers to traders at the beginning of every month. The top plot of figure 8.4 illustrates this periodicity. The lifespan of individual traders is tracked from May 15, 2000 to November 30, 2000. The x-axis represents the first trading day that a trader is active. The y-axis indicates the final trading day that a trader is active. Note that a subset of traders are observed to “survive” for more than one period. It was possible to resolve the reassignment of identifiers for certain traders by tracking limit orders placed before and transacted upon following the ID reassignment process. In the bottom plot of figure 8.4, the number of events occurring in each month is plotted. For May, data is available for only the latter half

[2]Information regarding the LSE data can be found at `http://www.londonstockexchange.com/en-gb/products/marketdata/`.

of the month and thus it has fewer events (approximately 18000). For most months, an average of 35000 events are seen. November is an anomaly with approximately 120000 events. For each month, from May to November of 2000, approximately 95 traders are observed. This value applies to the half of month of May as well as the event laden month of November.

Because of the periodicity caused by the trader ID reassignment process, it is not possible to examine periods of the Shell data exceeding one month using the trophic detection algorithm. Traders from different months[3] cannot possibly correlate with those of other months. Accordingly, traders can exhibit similarity only to those active in the same month. The trophic detection algorithm identifies these temporal delineations, and forms high-level communities accordingly. During the surrogate edge switching process, edges are rearranged such that traders of different months can share significant S_{ij} values. This redistribution destroys the community structure in the original network and thus the null test is easily rejected.

Results for LSE Shell data

Here, I examine the seven month-long data sets for Shell from May to November of 2000. Again, I test a range of E and W parameter settings. For each setting a battery of 10 analyses is conducted. The data for each analysis differs due to a random offset that determines the starting event. Because the month-long data sets provide a limited amount of data, I reduce the maximum random offset to 300 events.

Figure 8.5 presents results for the LSE Shell data sets with different W settings. For each parameter setting, two values are provided. Similar to the simulated data case, I provide the mean Z-scores[4] of D_o. However, unlike that case, it is not possible

[3]In this explanation, I exclude the traders for which the IDs have been resolved across multiple months.

[4]A Z-score reflects the distance from the mean (measured in units of standard deviation)

to provide the communal overlap values because there are no “known” communities. Instead, I provide the mean rank of the D_o value. Recall, to reject the null hypothesis, it is necessary for the original data to rank first in divisionality. The x-axis of each plot represents $W \in [25, 500]$. The left y-axis, associated with the blue triangular points, represents the mean Z-score of D_o. The right y-axis, associated with the red circular points, represents the mean rank of the D_o value.

The results presented in figure 8.5 are inconclusive. For the half month of May (top left), the null test is rejected for $W \geq 150$ and thus suggests the presence of significant community structure. The months of October (third row, right-side) and November (bottom left) suggest that community structure exists for particular ranges of W values. For October, the null test is consistently rejected for $325 \leq W \leq 475$ (and is rejected in the majority of analyses at $W = 500$). For November, the null hypothesis is consistently rejected within two ranges, $125 \leq W \leq 200$ and $400 \leq W \leq 500$. This discontinuity is surprising. For the months of June (top right) and September (third row, left-side), the values appear to fluctuate without strong patterns – thus, suggesting that significant community structure does not exist. The months of July (second row, left-side) and August (second row, right-side) are confounding. Here, the values are consistently low. Accordingly, the null test is never rejected. However, the consistency of these low values (especially for the month of August) is alarming. It suggests that the original network contains less community structure than expected in a randomly rewired version of itself. For each month, the number of low-level communities detected in the original networks falls within an approximate range of 31 to 34 with low variance. Accordingly, the fine-resolution communities have an approximate size of three traders.

that a particular value resides.

8.3 Discussion

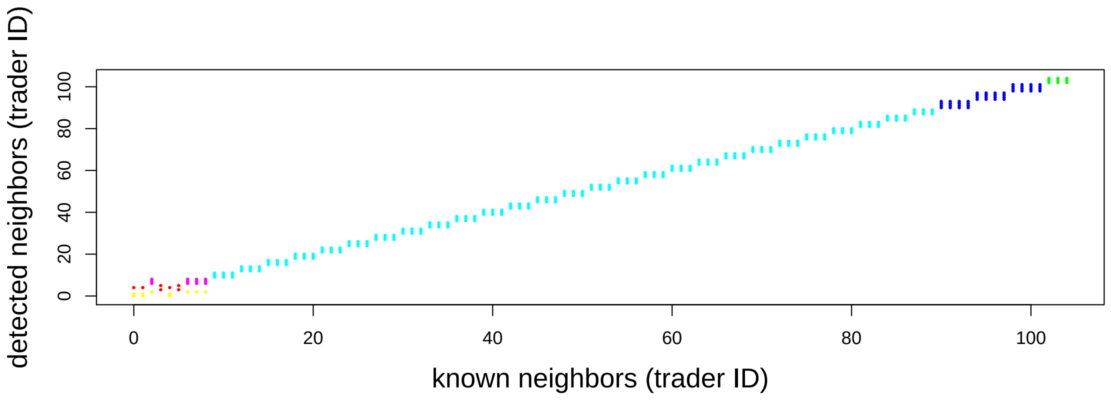
The results of section 8.2 show that it is possible to detect trophic structure in simulated data, where the behavior of traders is consistent. Results from the LSE data are less conclusive. For three of the seven months (May, October, and November), the null hypothesis is consistently rejected for significant ranges of W values. Two of these months include those with an anomalous number of events (May contains only 18000 events and November has 116155 events). The structure detected in the half-month of May suggests that perhaps shorter periods of data (measured in calendar time) should be examined. A possible explanation is that trader behaviors in the Shell data are not consistent for long periods of time. If so, to identify stable functional behavior would require that shorter periods (two weeks, possibly) be examined. Additionally, when significant structure is detected, the W values tend to be large. The use of even larger W values should also be investigated.

In this chapter, I attempted to detect trophic structure within disjointed, real-world data sets. However, assume that we have a continuous data set (one that does not suffer from the ID reassignment process of the LSE). Further, assume that we can successfully detect trophic structure in independent subsets of this data. Then, by comparing the trophic networks of overlapping subsets of data, we could study its evolution. Further, comparison of the changes in the trophic network to those of traditional market observables could improve our understanding of the dynamics responsible for stylized facts [22] commonly ascribed to financial markets.

I have focused on detecting the trophic species that comprise a trophic network. However, once detected, studying the interactions of these entities is a simple extension. For instance, by tracking the trades between trophic species, the flow of capital in a trophic network could be illustrated. Such a depiction is analogous to the food web of a biological system, where the transfer of energy is traced.

This chapter makes several contributions. First, the trader similarity measure of section 8.1.1 provides an extensible framework for the inclusion of multiple indicators. Further, in section 8.2.1, I demonstrated that use of the *buy/sell* indicators provides better detection of particular strategies (notably, market making) than does the *net trading* indicator employed by Zovko et al [65][5]. Further, in section 8.1.2, I propose a methodology for examining the significance of detected community structure. This approach differs from that of both Zovko et al [65] and Lillo et al [31]. In both of those studies, the significance of detected correlations (akin to my S_{ij} values) are examined using a null test based on random matrix theory [29]. However, demonstrating that these correlations are significant is different from showing that subsequent community structure based on these correlations is significant. In conclusion, although the LSE results are inconclusive, I believe that further examination of real-world financial market data is warranted.

[5]Lillo et al [31] employ a similar scheme. They investigate the *inventory variance* of traders. However, rather than representing the value with an enumerated indicator, they use the actual change in inventory measured in shares.



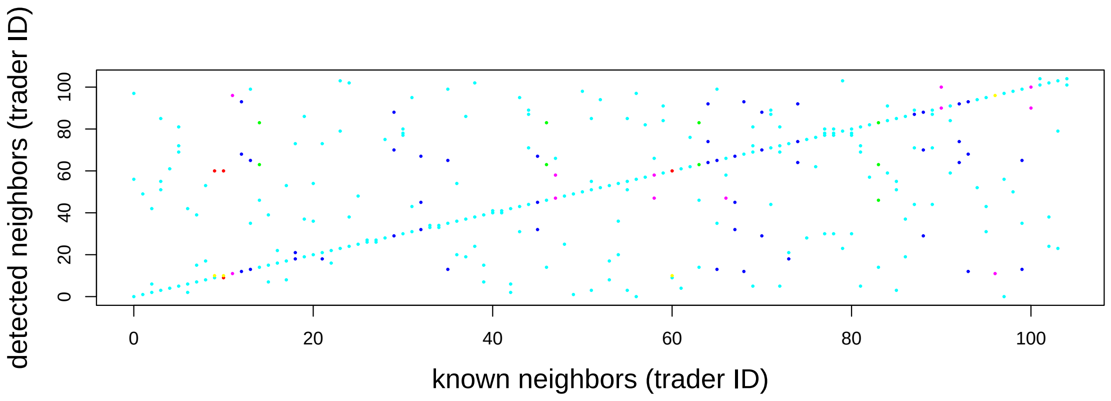
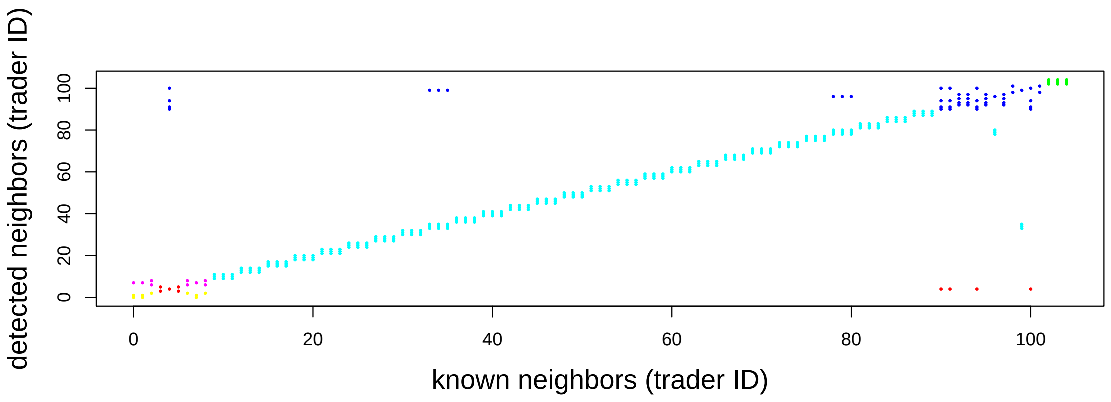


Figure 8.1: Two-dimensional trophic community plots for an analysis with $E = 20000$ and $W = 100$. Points on the x-axis of each plot represent the *known* trophic neighbors (by ID) for each trader (including itself). Points on the y-axis represent the IDs of the *detected* trophic neighbors for each trader. The top plot represents the original data and has an associated communal overlap value of 0.943. The bottom plot represents a surrogate data set that has an associated communal overlap value of 0.2.



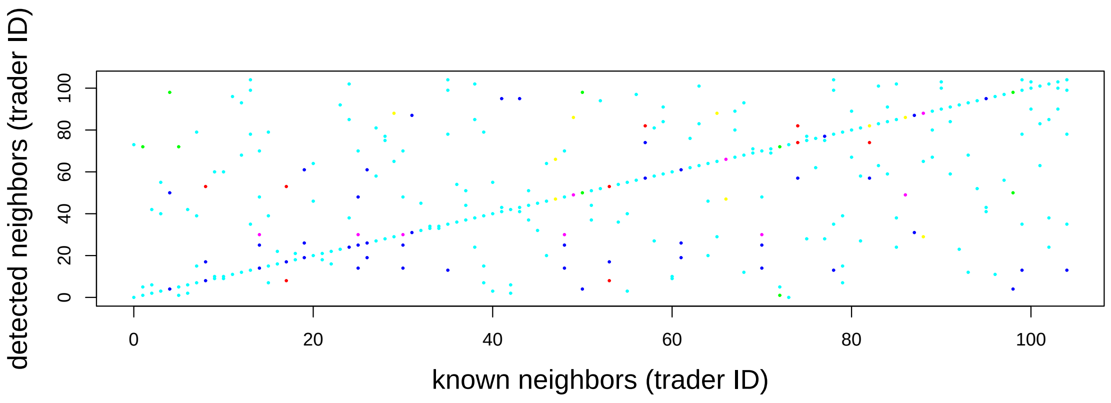


Figure 8.2: Two-dimensional trophic community plots for an analysis using a single indicator based on *net trading*. The data set employed is the same as that used to produce figure 8.1. Additionally, the same E and W settings are used. Points on the x-axis of each plot represent the *known* trophic neighbors (by ID) for each trader (including itself). Points on the y-axis represent the IDs of the *detected* trophic neighbors for each trader. The top plot represents the original data and has an associated communal overlap value of 0.849. The bottom plot represents a surrogate data set that has an associated communal overlap value of 0.241. Comparison of the top plot to the top plot of figure 8.1 shows that classification of the market makers (ranging from IDs 90-101) is less accurate using the *net trading* indicator instead of the *buy/sell* indicators.

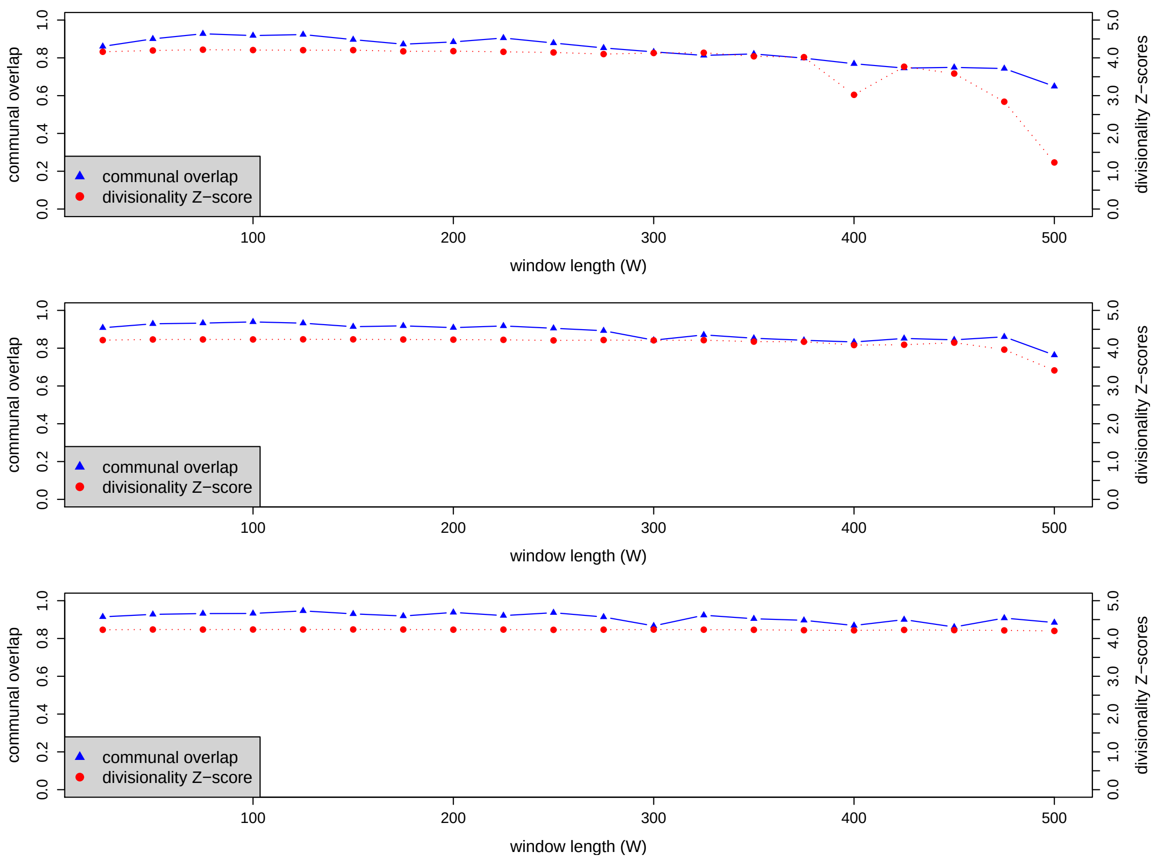
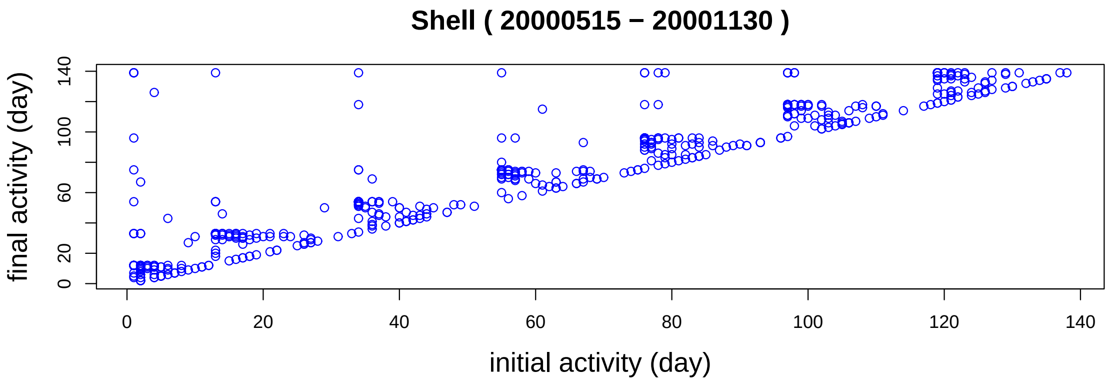


Figure 8.3: Results for the simulated data. The x-axis of each plot represents the time interval window length W. The top plot represents $E = 10000$, the middle plot represents $E = 20000$, and the bottom plot represents $E = 40000$. The left y-axis, associated with the triangular points, represents the mean communal overlap for 10 independent analyses with the specified parameter setting. The right y-axis, associated with the circular points, represents the mean Z-score of divisionality for the original data set.



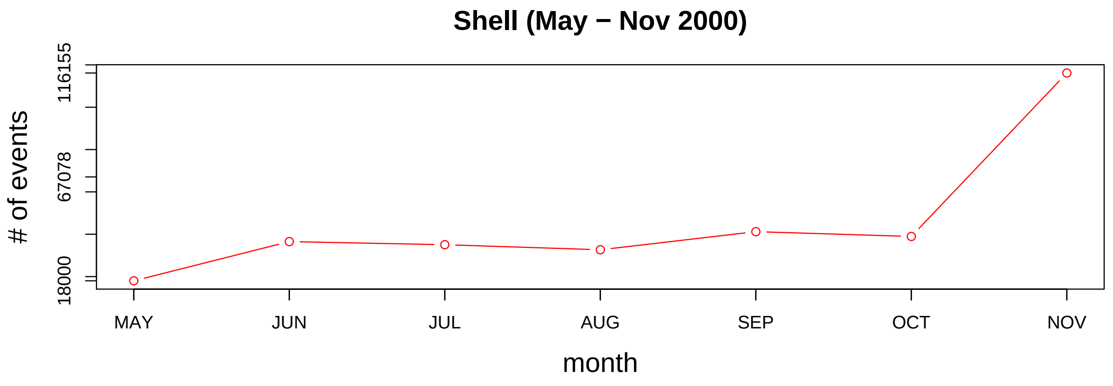


Figure 8.4: Lifespan of traders. In top plot, the lifespan of individual traders is tracked (from May 15, 2000 to November 30, 2000). The x-axis represents the first trading day that a trader is active. The y-axis indicates the final trading day that a trader is active. The periodicity is due to the trader ID reassignment process enforced by the LSE. Every month, traders are assigned new IDs. The points away from the diagonal represent traders for whom IDs were resolved despite the reassignment process. In the bottom plot, the number of events occurring in each month is plotted. For May, data is available for only the latter half of the month and thus it has fewer events (approximately 18000). For most months, an average of 35000 events are seen. November is an anomaly with approximately 120000 events.

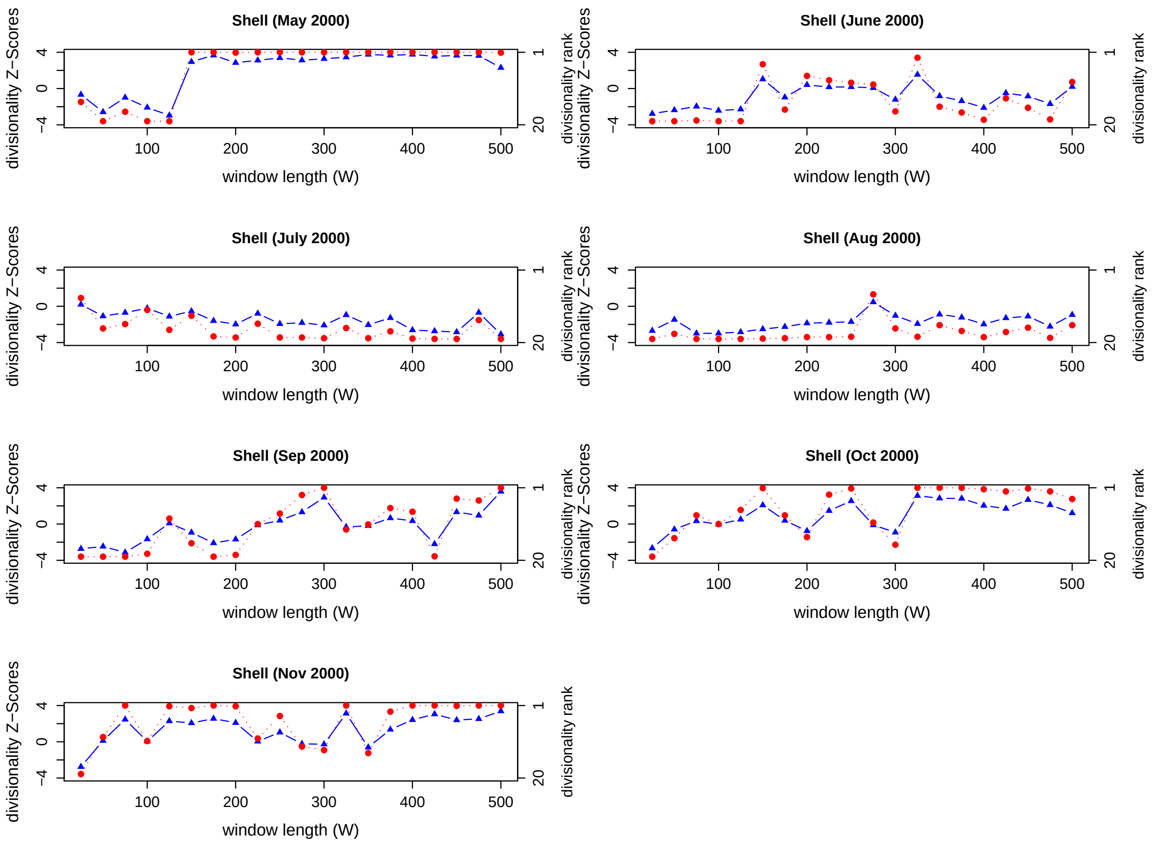


Figure 8.5: Results for the LSE Shell data. The months of May to October 2000 are independently examined. The x-axis of each plot represents $W \in [25, 500]$. The left y-axis, associated with the triangular points (blue), represents the mean Z-score of divisionality for the original data set. The right y-axis, associated with the circular points (red), represents the mean rank of the divisionality value for the original data set.

Chapter 9

Discussion & conclusions

Part I of this dissertation makes several contributions to the area of community detection in complex networks. In Chapter 3, I introduced a unique two-phase macro-strategy that uses induced fractures/merges of communities to improve detection. Used in conjunction with either existing community detection algorithms (micro-strategies) or my "quick and dirty" deterministic division (DD) algorithm, the macro-strategy provides high yield, robust results. As a consequence, practitioners no longer need to apply a given algorithm with multiple random number generator seeds to achieve confidence in a result. When the macro-strategy is run with DD, it achieves results that compete with the best from existing literature with reduced running times. A final advantage of the macro-strategy is that of incremental improvement, whereby progress of the algorithm can be tracked and terminated once an adequate result is achieved. In addition, it is possible to take an existing community detection result and improve upon it.

In Chapter 4, I demonstrated experimentally a resolution limit that is inherent to modularity, and I introduced a fine-granularity measure of community structure called divisionality. The December 2006 PNAS paper by Fortunato [18] illustrates the resolution limit through analysis. My research, conducted independently, was presented in a University of New Mexico Computer Science Department colloquium on September 15, 2006. In Chapter 8, the practical benefit of divisionality was demonstrated, allowing me to quantify the community structure in trader networks at a fine resolution. It is possible to isolate low-level community structure by recursively optimizing modularity on sub-networks of a trader network. However, as a statistical measure, modularity does not quantify the fine-granularity structure of the global network. Divisionality provides a solution.

The dual-assortative measure (DAMM) of community structure presented in Chapter 5 extends the domain of problems for which community detection algorithms can be applied. With DAMM, it is possible to analyze either networks containing solely negative edges or networks containing both positive and negative edges. Previously, only networks with positive edges could be analyzed. As an example of its practical efficacy, consider the work of Zovko and Farmer [65], where correlation matrices based on the similarity of trading behaviors are analyzed using a clustering algorithm. The DAMM allows for an analogous study based on community structure[1], rather than clustering. Previous to DAMM, valuable information conveyed by negative correlations was ignored. Using DAMM, both positive and negative correlations contribute to the detection of structure.

The research in Part II is rooted to the beginning stages of my doctorate. As a member of the Adaptive Computation Laboratory at the University of New Mexico, my initial interest was to extend the paradigm of genetic programming (GP). Construction of the Staq language preceded my interest in financial markets. When I began working with Doyne Farmer at the Santa Fe Institute, I viewed financial markets as an arena in which to test GP. My initial objective was to coevolve a

[1]First, a network must be constructed. Nodes would represent independent traders. Weighted edges would represent the correlations. Then, a community detection algorithm would be applied to the network.

population of traders that generated dynamics approximating those observed in real financial markets. To confirm these characteristics, I planned to construct a *market Turing test*[2]. A market simulator that could pass such a test would allow the modern economist to conduct controlled experiments, and thus, provide an invaluable tool for the study of macro dynamics in financial markets.

In hindsight, I believe that the work presented in part II suffers from an identity crisis. I was torn between my desire to extend GP and simultaneously wanting to contribute to the study of financial markets[3]. If I were to start the endeavor over, I would focus solely on one of the objectives rather than both. With regards to studying financial markets, I would adopt a more constrained agent-based approach than that of evolving traders with genetic programming. The software development of Staq, the market simulator, and the subsequent parallelization of each code-base posed an enormous engineering undertaking[4]. Another prominent challenge introduced by my GP approach was interpreting the Staq code of an evolved trader. Visual inspection of an evolved Staq program exceeding twenty tokens can prove vexing due to the preponderance of *junk DNA*[5]. To ascertain the functionality of a Staq trader, it was necessary to build additional software visualization tools.

[2]The idea of the test is inspired by the famous *Turing test* [58], which was proposed to determine whether a machine could perform a human-like conversation. In the market version, two data streams are presented. One stream originates from a real world market; the other is generated by a market simulator. If a judge cannot distinguish which data stream represents the real market, the market simulator is declared to have passed the test.

[3]I was forewarned by my advisors Stephanie Forrest and Doyne Farmer, but was seduced by the sirens of evolution.

[4]The code-base exceeds 60000 lines of code. The software was parallelized to run on a compute farm at the University of New Mexico Center for High Performance Computing.

[5]Here, I refer to code fragments that provide useful functionality on their own, but cannot be reached during execution, as *junk DNA*. One of the most interesting things that I learned with Staq was that the evolutionary agents seemed to develop a *library* of code. Any given agent would utilize a fraction of this library code. However, by carrying *junk*, the unused code remained in the gene pool and could be utilized by later generations. Each junk code fragment was likely incorporated into the gene pool by an ancestor for which it performed a useful task. In a sense, the junk code provides an evolutionary history.

Having stated these limitations, the evolved traders of Chapter 6 demonstrated the efficacy of Staq. The clear & hoist (CH) strategy, though not useful in a real-world setting, clearly suggests that the evolutionary process identifies limitations in the simulated market environment. Further, after adjusting the fitness function to account for inventory control, the real-world strategy of market making consistently evolves. To evolve more interesting evolutionary solutions, it is necessary to either alter the simulated market environment or transition to a coevolutionary scheme. Sophisticated strategies are not required to profit from the zero-intelligence noise traders. Thus, selective pressure does not exist for them to develop. Further, given the employed terminal set, market making strategies can be represented compactly. As such, they present a strong attractor for the evolutionary process.

In Chapter 7, I addressed a limitation revealed by the CH trader: a lack of fundamentalism in the simulated market environment. To remedy the shortcoming, I introduced two value-based trading strategies for the continuous double auction (CDA) and demonstrated that both successfully encourage price to track value. Further, both strategies neutralize the CH trader by interfering with its ability to inflate prices. Accordingly, in a coevolutionary scheme, the presence of a value-based trader would serve a valuable role. Further, it raises an interesting question: would value-based processes evolve intrinsically with a coevolutionary process?

Although Part II concentrates on the behavior of individual agents, I reiterate the initial, and ultimate, goal: to coevolve a population of traders. By allowing multiple GP agents to compete against one another, evolutionary selective pressure will be more realistic. As a result, the evolution of more sophisticated trader strategies – and more meaningful economic conclusions – may follow. Due to a combination of factors – chiefly, the enormous search space resulting from the employed GP terminal set and computational bottlenecks – progress on the coevolutionary front was slower than anticipated.

In Part III, I demonstrated that the tools developed in Part I can be used to detect trophic species in simulated financial market data. This chapter provides several useful contributions. First, in section 8.2.1, I show that use of independent *buy/sell* indicators, as opposed to the *net trading* indicator [31] [65], improves detection of market making strategies. Further, the *trader similarity* measure provides an extensible framework for assessing multiple indicators. It is built upon the information theoretic measure of symmetric uncertainty. However, by utilizing DAMM, Spearman rank correlation could be used to measure the similarity of indicator sets, instead. Finally, the use of a rank-based test and surrogate data provides a methodology for testing the significance of detected structure. Examining real-world data from the London Stock Exchange (LSE) provided inconclusive results. However, more extensive investigation of real-world data – preferably, data that does not suffer from the trader identifier reassignment limitation present in the LSE data – is needed to better determine the efficacy of the approach. I believe that further research is warranted.

Appendix A

Appendix for *Community detection optimization with induced fractures and merges*

A.1 Calculating ΔQ^u and $\Delta^2 Q^{um}$

In a divisive scheme, ΔQ^u measures the change in modularity that would result by migrating node u from one community to the other. Equation A.1 presents the calculation:

$$\Delta Q^u \quad = \quad 2\left[\left(\frac{w_{gu} - w_{lu}}{T}\right) + \frac{d_u}{T^2}\left(a_l - a_g - d_u\right)\right] \tag{A.1}$$

.

Here, the community losing the node u is denoted with an l and the community gaining the node is denoted with a g. Accordingly, w_{gu} represents the cumulative edge weight between node u and the gain community, w_{lu} represents the cumulative edge weight between node u and the loss community, d_u represents the degree of node u, a_l represents the cumulative edge weight of the loss community, and a_g represents the cumulative edge weight of the gain community. T represents twice the total edge weight of the network such that $T = \sum_i \sum_j w_{ij}$, where w_{ij} represents the cumulative edge weight between community i and community j.

Following the migration of a node, each of the ΔQ^u values is subject to change. Rather than recalculating each value using equation A.1, it is more computationally efficient to adjust each value by adding $\Delta^2 Q^{um}$, which represents the change in ΔQ^u that results from migrating a node m. We calculate $\Delta^2 Q^{um}$ as:

$$\Delta^2 Q^{um} = 4D\frac{-d_u d_m}{T^2} + \frac{w_{mu}}{T} \qquad \text{(A.2)}$$

where $D \in \{-1, 1\}$ represents a direction indicator. If the nodes m and u migrate in the same direction, $D = 1$. Otherwise, $D = -1$. The w_{mu} term represents the edge weight between the nodes m and u.

A.2 Divisive results

Table A.1 presents the raw modularity statistics for the macro-strategy using EO. Table A.2 presents the same for the macro-strategy using the DD algorithm.

Table A.1: Results for extremal optimization.

network	phase 1 mean(Q)	phase 1 sd(Q)	phase 2 mean(Q)	phase 2 sd(Q)	phase 2 max(Q)
karate	0.4125	0.00999	0.4198	0.00000	0.4198
jazz	0.4394	0.00505	0.4450	0.00020	0.4451
celegans	0.4227	0.00752	0.4533	0.00192	0.4561
email	0.5521	0.01075	0.5806	0.00163	0.5827
pgp	0.8359	0.00521	0.8745	0.00103	0.8769

Table A.2: Results for deterministic division.

network	phase 1 mean(Q)	phase 2 mean(Q)	phase 2 sd(Q)	phase 2 max(Q)
karate	0.3965	0.4198	0.0000	0.4198
jazz	0.4344	0.4450	0.0003	0.4451
celegans	0.3874	0.4472	0.0030	0.4519
email	0.5151	0.5775	0.0017	0.5807
pgp	0.7591	0.8441	0.0033	0.8512

A.3 Agglomerative results

Table A.3 presents the raw modularity statistics for the macro-strategy using the CNM agglomeration algorithm.

Table A.3: Results for the CNM agglomerative algorithm.

network	phase 1 mean(Q)	phase 2 mean(Q)	phase 2 sd(Q)	phase 2 max(Q)
karate	0.3807	0.4195	0.0009	0.4198
jazz	0.4389	0.4445	0.0006	0.4449
celegans	0.4069	0.4484	0.0041	0.4533
email	0.5044	0.5738	0.0028	0.5777
pgp	0.8496	0.8624	0.0038	0.8699

A.4 Evolution of community detection algorithms

Results printed in table A.4 compare the best results of several micro-strategies from existing literature.

Table A.4: Best modularity results from existing literature.

		modularity			
network	size n	GN	CNM	DA	spectral
karate	34	0.401	0.381	0.4188	0.419
jazz	198	0.405	0.439	0.4452	0.442
metabolic	453	0.403	0.402	0.4342	0.435
email	1133	0.532	0.494	0.5738	0.572
pgp	10680	0.816	0.733	0.8459	0.855
condmat	27519	-	0.668	0.6790	0.723

Appendix B

Appendix for *A dual assortative measure of community structure*

B.1 Derivation of $\Delta Q^{D,u}$

As expressed in equation 5.5, the DAMM, Q^D, involves summing the independent contributions of all communities i. We can represent the contribution a given community i with a local DAMM value, denoted as q_i, such that $Q^D = \sum_i q_i$. Further, if we wish to independently assess the positive and negative edge weight contributions of each given community, we can write $Q^D = \sum_i q_i^+ + q_i^-$.

Computing $\Delta Q^{D,u}$ entails comparing the DAMM value from timestep t to the DAMM value at time $t+1$ that results from migrating node u. We denote the DAMM value at time t as Q_t^D and the DAMM value following the migration of node u as $Q_{t+1(u)}^D$ Accordingly, $\Delta Q_t^{D,u}$ can be expressed as:

$$\Delta Q_t^{D,u} \quad = \quad Q_{t+1(u)}^D - Q_t^D \tag{B.1}$$

.

Utilizing the local DAMM notation, we can rewrite equation B.1 as:

$$\Delta Q_t^{D,u} = (\sum_i q^+_{i,t+1(u)} + q^-_{i,t+1(u)}) - (\sum_i q^+_{i,t} + q^-_{i,t}) \tag{B.2}$$

where $q^+_{i,t}$ represents the positive edge contribution of the local DAMM value for community i at time t (before migrating the node u) and $q^+_{i,t+1(u)}$ represents the positive edge contribution of the same community following the migration of node u. The subscript $t+1(u)$ is used to indicate that a node u has been migrated.

By grouping the positive local DAMM values and the negative local DAMM values separately, we can write:

$$\begin{aligned} \Delta Q_t^{D,u} &= \sum_i (q^+_{i,t+1(u)} - q^+_{i,t}) + \sum_i (q^-_{i,t+1(u)} - q^-_{i,t}) && \text{(B.3)} \\ &= \Delta Q_t^{+,u} + \Delta Q_t^{-,u} && \text{(B.4)} \end{aligned}$$

.

By moving a single vertex from one community to another, as is done during the division process, only two communities are affected. All other communities remain unchanged. One community gains a new node. We refer to this community as the *gain* community, and denote it as C_g. Conversely, the other affected community loses a node. We denote this *loss* community as C_l. Since only the C_g and C_l communities are affected by a vertex move, the only local DAMM values that change are those relating to these communities – q_g and q_l. The local DAMM contributions of all other communities, $\{q_i | i \neq g, i \neq l\}$, remain unchanged. Thus, we need only to assess the change in DAMM for the gain and loss communities. Using this information, we rewrite $\Delta Q_t^{+,u}$ as:

$$\begin{aligned}
\Delta Q_t^{+,u} &= \sum_i q^+_{i,t+1(u)} - q^+_{i,t} && \text{(B.5)}\\
&= (q^+_{g,t+1(u)} - q^+_{g,t}) + (q^+_{l,t+1(u)} - q^+_{l,t}) && \text{(B.6)}\\
&= \Delta q_{g,t}^{+,u} + \Delta q_{l,t}^{+,u} && \text{(B.7)}
\end{aligned}$$

where $q^+_{g,t}$ denotes the local DAMM value for the positive edges associated with the gain community at time t, $q^+_{l,t}$ denotes the local DAMM value for the positive edges of the loss community at time t, and $q^+_{g,t+1(u)}$ represents the local DAMM value for the positive edge contributions of the gain community following the migration of node u.

We can now separately analyze the contributions of $\Delta q_{g,t}^{+,u}$ and $\Delta q_{l,t}^{+,u}$ and reassemble the terms to establish $\Delta Q_t^{+,u}$. We denote the node to be moved as u and introduce the following notation: $\{w_{gg} = \sum_{rs} e_{rs} | r \in C_g, s \in C_g\}$ to represent the cumulative intra-community positive edge weight for the gain community, $\{w_{gu} = \sum_{rs} e_{rs} | r \in C_g, s = u\}$ to represent the cumulative edge weight of the gain community connected to node u, and d_u to represent the positive degree of node u. Note that $q^+_{g,t}$, which represents the contribution of the positive edges in the gain community prior to moving the node, is defined as:

$$q^+_{g,t} = \frac{2w_{gg}}{T} - \left(\frac{a_g}{T}\right)^2 \tag{B.8}$$

.

We use the notation $q^{+}_{g,t+1(u)}$ to denote the contribution of the gain community positive edges after migrating node u. Following the migration, the gain community contains an additional node. Accordingly, both the cumulative intra-community positive edge weight, w_{gg}, and the total cumulative positive edge weight of the gain community, a_g, are subject to change. More specifically, the intra-community positive edge weight is updated as $w_{gg,t+1(u)} = w_{gg,t} + w_{gu,t}$, where $w_{gu,t}$ represents the cumulative positive edge weight connecting the node u to the gain community prior to the migration. Furthermore, the cumulative positive edge weight of the gain community is updated as $a_{g,t+1(u)} = a_{g,t} + d_u$. Using these updates, we define $q^{+}_{g,t+1(u)}$ as:

$$q^{+}_{g,t+1(u)} = \frac{2(w_{gg} + w_{gu})}{T} - \left(\frac{a_g + d_u}{T}\right)^2 \tag{B.9}$$

.

By subtracting equation B.8 from equation B.9, we establish $\Delta q^{+,u}_{g,t}$ as provided in equation B.11:

$$\begin{aligned} \Delta q^{+,u}_{g,t} &= q^{+}_{g,t+1(u)} - q^{+}_{g,t} && \text{(B.10)} \\ &= \frac{2w_{gu}}{T} - \frac{2d_u}{T^2}\left(a_g + \frac{d_u}{2}\right) && \text{(B.11)} \end{aligned}$$

Similarly, the positive edge contribution of the loss group, denoted as $\Delta q^{+,u}_{l,t}$, is defined by equation B.14.

$$
\begin{aligned}
\Delta q_{l,t}^{+,u} &= q_{l,t+1(u)}^{+} - q_{l,t}^{+} && \text{(B.12)}\\
&= \left[\frac{2(w_{ll}-w_{lu})}{T} - \left(\frac{a_l - d_u}{T}\right)^2\right] \\
&\quad -\left[\frac{2w_{ll}}{T} - \left(\frac{a_l}{T}\right)^2\right] && \text{(B.13)}\\
&= \frac{-2w_{lu}}{T} + \frac{2d_u}{T^2}\left(a_l - \frac{d_u}{2}\right) && \text{(B.14)}
\end{aligned}
$$

By assembling equations B.11 and B.14, we establish $\Delta Q_t^{+,u}$ as provided by equation B.16.

$$
\begin{aligned}
\Delta Q_t^{+,u} &= \Delta q_{g,t}^{+,u} + \Delta q_{l,t}^{+,u} && \text{(B.15)}\\
&= 2\left[\left(\frac{w_{gu} - w_{lu}}{T}\right) + \frac{d_u}{T^2}\left(a_l - a_g - d_u\right)\right] && \text{(B.16)}
\end{aligned}
$$

Following a similar logic, we establish $\Delta Q_t^{-,u}$. For brevity, we provide the result in equation B.18, where $\bar{w}_{gu}$ represents the cumulative negative edge weight of the gain community connected to node u and $\bar{d}_u$ represents the negative degree of the node u.

$$
\begin{aligned}
\Delta Q_t^{-,u} &= \Delta q_{g,t}^{-,u} + \Delta q_{l,t}^{-,u} && \text{(B.17)}\\
&= 2\left[\left(\frac{\bar{w}_{lu} - \bar{w}_{gu}}{T}\right) + \frac{\bar{d}_u}{T^2}\left(\bar{a}_l - \bar{a}_g - \bar{d}_u\right)\right] && \text{(B.18)}
\end{aligned}
$$

Finally, we establish $\Delta Q_t^{D,u}$ by assembling equations B.16 and B.18:

$$\begin{aligned}
\Delta Q_t^{D,u} &= \Delta Q_t^{+,u} + \Delta Q_t^{-,u} && \text{(B.19)}\\
&= 2\left[\left(\frac{w_{gu}-w_{lu}}{T}\right) + \frac{d_u}{T^2}\left(a_l - a_g - d_u\right)\right]\\
&\quad +2\left[\left(\frac{\bar{w}_{lu}-\bar{w}_{gu}}{T}\right) + \frac{\bar{d}_u}{T^2}\left(\bar{a}_g - \bar{a}_l + \bar{d}_u\right)\right] && \text{(B.20)}
\end{aligned}$$

B.2 Derivation of $\Delta^2 Q^{D,um}$

As a first step of our derivation, we concentrate solely on the contributions of the positive edges. First, we analyze the difference $\Delta^2 Q_t^{+,um} = \Delta Q_{t+1(m)}^{+,u} - \Delta Q_t^{+,u}$, where $\Delta Q_{t+1(m)}^{+,u}$ represents the change in DAMM that would result from the migration of node u if node m were already to have been migrated given the current configuration. $\Delta Q_t^{+,u}$, which pertains exclusively to positive edges, was provided in equation B.16. To simplify the derivation, we independently assess the first and second terms of equation B.16, such that $A_t^u = \frac{w_{gu}-w_{lu}}{T}$ and $B_t^u = (a_l - a_g - d_u)$.

After migrating node m, the value of A may be altered – and, this change should be reflected in $A_{t+1(m)}^u$. Note the two terms involved in A_t^u: w_{gu} and w_{lu}. Each term measures the cumulative positive edge weight connecting a community – either the gain or loss community – to the node u prior to the migration of node m. The migration of node m may or may not alter the cumulative positive edge weight between each community and node u. If node m was migrated to the community currently not occupied by u, the *gain* community, w_{gu} will increase such that $w_{gu,t+1} = w_{gu,t} + e_{mu}$, where e_{mu} represents the positive edge weight between the m and u nodes. Otherwise, if the m node moved in the opposite direction, $w_{gu,t+1(m)} = w_{gu,t} - e_{mu}$. Using the direction indicator D, we can write $w_{gu,t+1(m)} = w_{gu,t} + De_{mu}$. Similarly, we can write $w_{lu,t+1(m)} = w_{lu,t} - De_{mu}$. By assembling the two terms, we express $A_{t+1(m)}^u$ as:

$$A^u_{t+1(m)} = \frac{(w_{gu} + De_{mu}) - (w_{lu} - De_{mu})}{T} \tag{B.21}$$

.

By subtracting A^u_t from $A^u_{t+1(m)}$, we establish ΔA^u_t as shown in equation B.24.

$$\begin{aligned}\Delta A^u_t &= A^u_{t+1(m)} - A^u_t &\text{(B.22)}\\ &= \frac{(w_{gu} + De_{mu}) - (w_{lu} - De_{mu})}{T} \\ &\quad - \frac{(w_{gu} - w_{lu})}{T} &\text{(B.23)}\\ &= \frac{2De_{mu}}{T} &\text{(B.24)}\end{aligned}$$

.

The first two terms of B, a_l and a_g, are similarly affected by the migration of node m. These terms represent the cumulative positive edge weight of the loss and gain communities, respectively. The updated terms can be expressed as $a_{l,t+1(m)} = a_{l,t} - Dd_m$ and $a_{g,t+1(m)} = a_{g,t} + Dd_m$, where d_m represents the positive degree of the m node. Accordingly, we can write $B^u_{t+1(m)}$ as:

$$B^u_{t+1(m)} = (a_l - Dd_m) - (a_g + Dd_m) - d_u \tag{B.25}$$

.

By subtracting B^u_t from $B^u_{t+1(m)}$, we establish ΔB^u_t as seen in equation B.28.

$$\begin{aligned}
\Delta B_t^u &= B_{t+1(m)}^u - B_t^u && \text{(B.26)}\\
&= [(a_l - Dd_m) - (a_g + Dd_m) - d_u]\\
&\quad -(a_l - a_g - d_u) && \text{(B.27)}\\
&= -2Dd_m && \text{(B.28)}
\end{aligned}$$

We now utilize ΔA_t^u and ΔB_t^u to establish the positive edge contribution to $\Delta^2 Q^{D,um}$:

$$\begin{aligned}
\Delta^2 Q_t^{+,um} &= 2\left[\Delta A_t^u + \frac{d_u}{T^2}\Delta B_t^u\right] && \text{(B.29)}\\
&= 2\left[\frac{2De_{mu}}{T} + \frac{d_u}{T^2}(-2Dd_m)\right] && \text{(B.30)}\\
&= 4D\left[\frac{e_{mu}}{T} - \frac{d_u d_m}{T^2}\right] && \text{(B.31)}
\end{aligned}$$

.

Following a similar approach, it is possible to establish the negative edge contribution to $\Delta^2 Q^{D,um}$:

$$\Delta^2 Q_t^{-,um} = 4D\left[\frac{\bar{d}_u \bar{d}_m}{T^2} + \frac{\bar{e}_{mu}}{T}\right] \quad \text{(B.32)}$$

.

By assembling $\Delta^2 Q_t^{+,um}$ of equation B.31 and $\Delta^2 Q_t^{-,um}$ of equation B.32, we establish the generalized equation:

$$\begin{aligned}
\Delta^2 Q_t^{D,um} &= \Delta^2 Q_t^{+,um} + \Delta^2 Q_t^{-,um} && \text{(B.33)}\\
&= 4D\left[\frac{(\bar{d}_u\bar{d}_m - d_u d_m)}{T^2} + \frac{(e_{mu} + \bar{e}_{mu})}{T}\right] && \text{(B.34)}
\end{aligned}$$

.

References

[1] Martin Andrews and Richard Prager. *Advances in genetic programming*, chapter Genetic programming for the acquisition of double auction markets strategies, pages 355 – 368. MIT Press, 1994.

[2] Alex Arenas, Alberto Fernandez, and Sergio Gomez. Analysis of the structure of complex networks at different resolution levels. *arXiv:physics/0703218v2*, 2007.

[3] W. Brian Arthur, John H. Holland, Blake LeBaron, R.G. Palmer, and P. Tayler. Asset pricing under endogenous expectations in an artificial stock market. In *The Economy as an Evolving Complex System II*, pages 15–44. Addison-Wesley, 1997.

[4] N. Bansal, A. Blum, and S. Chawla. Correlation clustering, 2002.

[5] Fischer Black. Noise. *Journal of Finance*, 41, 1986.

[6] S. Boettcher and A. G. Percus. Extremal optimization for graph partitioning. *Physical Review E*, 64, 2001.

[7] S. Boettcher and A. G. Percus. Optimization with extremal dynamics. *Physical Review Letters*, 86:5211–5214, 2001.

[8] U. Brandes, D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, and D. Wagner. Maximizing modularity is hard. Technical report, arXiv:physics/0608255, 2006.

[9] Carl Chiarella and Giulia Iori. A simulation analysis of the microstructure of double auction markets. *Quantitative Finance*, 2:346–353, 2002.

[10] Carl Chiarella and Giulia Iori. The impact of heterogeneous trading rules on the limit order book and order flows. Research Paper Series 152, Quantitative Finance Research Centre, University of Technology, Sydney, February 2005. available at http://ideas.repec.org/p/uts/rpaper/152.html.

[11] A. Clauset, M.E.J. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E*, 70, 2004.

[12] D. Cliff. Minimal-intelligence agents for bargaining behaviors in market-based environments, 1997.

[13] Marcus G. Daniels, J. Doyne Farmer, Giulia Iori, and Eric Smith. How storing supply and demand affects price diffusion. arXiv:cond-mat/0112422v5, January 2002.

[14] Jordi Duch and Alex Arenas. Community detection in complex networks using extremal optimization. *Phys. Rev. E*, 2005.

[15] J.D. Farmer. Market force, ecology and evolution. *Ind. & Corp. Change*, 11:895–953, 2002.

[16] J.D. Farmer, L. Gillemot, G. Iori, S. Krishnamurthy, D. E. Smith, and M. G. Daniels. *The Economy as an Evolving Complex System, III*, chapter A Random Order Placement Model of Price Formation in the Continuous Double Auction, pages 133–173. Oxford University Press, 2005.

[17] J.D. Farmer, P. Patelli, and I. Zovko. The predicitive power of zero intelligence in financial markets. *PNAS*, 102:2254–2259, 2005.

[18] Santo Fortunato and Marc Barthelemy. Resolution limit in community detection. arXiv. http://arxiv.org/abs/physics/0607100.

[19] M. Girvan and M.E. J. Newman. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA*, 99:7821–7826, 2002.

[20] D.K. Gode and S. Sunder. Allocative efficiency of markets with zero-intelligence traders: market as a partial substitute for individual rationality. *The Journal of Political Economy*, 101(1):119–137, 1993.

[21] R. Guimera and L.A.N. Amaral. *J Stat Mech*, 2005.

[22] N.F. Johnson, P. Jefferies, and P.M. Hui. Physics and finance: Coursebook. Used for Oxford University course entitled *Econophsics*, 2002.

[23] Michael Kearns and Luis Ortiz. The penn-lehman automated trading project. *IEEE Intelligent Systems*, Nov/Dec, 2003.

[24] B.W. Kernighan and S. Lin. A efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49:291–307, 1970.

[25] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, 1994.

[26] John R. Koza. Genetic programming. In James G. Williams and Allen Kent, editors, *Encyclopedia of Computer Science and Technology*, volume 39, pages 29–43. Marcel-Dekker, 1998.

[27] John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann, 1999.

[28] J.M. Kumpula, J. Saramaki, K. Kaski, and J. Kertesz. Resolution limit in complex network community detection with potts model approach. *Eur. Phys. J.*, 56, 2007.

[29] L. Laloux, P. Cizeau, J.P. Bouchard, and M. Potters. Noise dressing of financial correlation matrices. *Physical Review Letters*, 88:1467–1470, 1999.

[30] E.A. Leicht and M.E.J. Newman. Community structure in directed networks. *Physical Review Letters*, in press.

[31] Fabrizio Lillo, Esteban Moro, and Rosario Mantegna. Ecology of trading firms in a financial market. Complexity 2006.

[32] Michael J. Mauboussin. The stock market as a complex adaptive system. *Journal Of Applied Corporate Finance*, 14, No 4, 2002.

[33] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, 69, 2004.

[34] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69, 2004.

[35] M.E.J. Newman. Assortative mixing in networks. *Phys. Rev. Lett.*, 89, 2002.

[36] M.E.J. Newman. Mixing patterns in networks. *Phys. Rev. E*, 67, 2003.

[37] M.E.J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[38] M.E.J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, 74, 2006.

[39] M.E.J. Newman. Modularity and community structure in networks. *Proc. Natl. Acad. Sci.*, 103:8577–8582, 2006.

[40] M.E.J. Newman and M. Girvan. Mixing patterns and community structure in networks. *Statistical Mechanics of Complex Networks*, pages 66–87, 2003.

[41] R.G. Palmer, W. Brian Arthur, John H. Holland, and Blake LeBaron. An artificial stock market. In *Artificial Life and Robotics*, 1998.

[42] R.G. Palmer, W. Brian Arthur, John H. Holland, Blake LeBaron, and P. Taylor. Artificial economic life: a simple model of a stock market. *Physica D*, 75:264–274, 1994.

[43] S. Phelps, S. Parsons, P. McBurney, and E. Sklar. Coevolution of auction mechanisms and trading strategies: Towards a novel approach to microeconomic design, 2002.

[44] Josep Pujol, Javier Bejar, and Jordi Delgado. Clustering algorithm for determining community structure in large networks. *Phys Rev E*, 74, 2007.

[45] J. Reichardt and S. Bornholdt. Detecting fuzzy community structures in complex networks with a potts model. *Physical Review Letters*, 93, 2004.

[46] J. Rust, J. Miller, and R. Palmer. *The Double Auction Market: Institutions, Theories, and Evidence*, chapter Behavior of trading automata in a computerized double auction market, pages 155–198. Addison-Wesley, 1992.

[47] J. Rust, J. Miller, and R. Palmer. Characterizing effective trading strategies. *Journal of Economic Dynamics and Control*, 18:61–96, 1994.

[48] W.P. Salman, O. Tisserand, and B. Toulot. *Forth*. Springer-Verlag:Berlin, 1984.

[49] J. Scott. *Social Network Analysis: A Handbook*. Sage Publications, 2000.

[50] William F. Sharpe. The sharpe ratio. *The Journal of Portfolio Management*, Fall, 1994.

[51] Andrei Shleifer and Lawrence Summers. The noise trader approach to finance. *The Journal of Economic Perspective*, 4:19–33, 1990.

[52] Eric Smith, J. Doyne Farmer, Laszlo Gillemot, and Supriya Krishnamurthy. Statistical theory of the continuous double auction. *Quantitative Finance*, 3(6):481–514, 2003.

[53] V.L. Smith. An experimental study of competitve market behavior. *Journal of Political Economy*, 70:111–137, 1962.

[54] V.L. Smith and A. Williams. Experimental market economics. *Scientific American*, 267(6):72–77, 1992.

[55] L. Spector and A. Robinson. Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3:7–40, 2002.

[56] Peter Stone, Michael L. Littman, Satinder Singh, and Michael Kearns. ATTac-2000: an adaptive autonomous bidding agent. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 238–245, Montreal, Canada, 2001. ACM Press.

[57] J. Thelier, B. Galdrikian, A. Longtin, S. Eubank, and J.D. Farmer. Using surrogate data to detect nonlinearity in time series. In M. Casdagli and S. Eubank, editors, *Nonlinear Modelling and Forecasting*, pages 163–188. Addison Wesley, 1992.

[58] Alan Turing. Computing machinery and intelligence. *Mind*, LIX, no. 236:433–460, 1950.

[59] S. Wasserman and K. Faust. *Social Network Analysis.* Cambridge University Press, 1994.

[60] M. Wellman, P. Wurman, K. O'Malley, R. Bangera, S. Lin, D. Reeves, and W. Walsh. Designing the market game for a trading agent competition. *IEEE Internet Computing*, 5:43–51, 2001.

[61] M. Wellman, P. Wurman, K. O'Malley, R. Bangera, S. Lin, D. Reeves, and W. Walsh. The 2001 trading agent competition. *Electronic Markets*, 13:4–12, 2003.

[62] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4:65–85, 1994.

[63] Darrell Whitley, Soraya Rana, and Robert B. Heckendorn. The island model genetic algorithm: On separability, population size and convergence. citeseer.ist.psu.edu/whitley98island.html.

[64] Ian H. Witten and Eibe Frank. *Data Mining.* Academic Press, 2005.

[65] Ilija Zovko and J.D. Farmer. Correlations and clustering in the trading of members of the london stock exchange. *Santa Fe Institute Working Paper*, 2007.